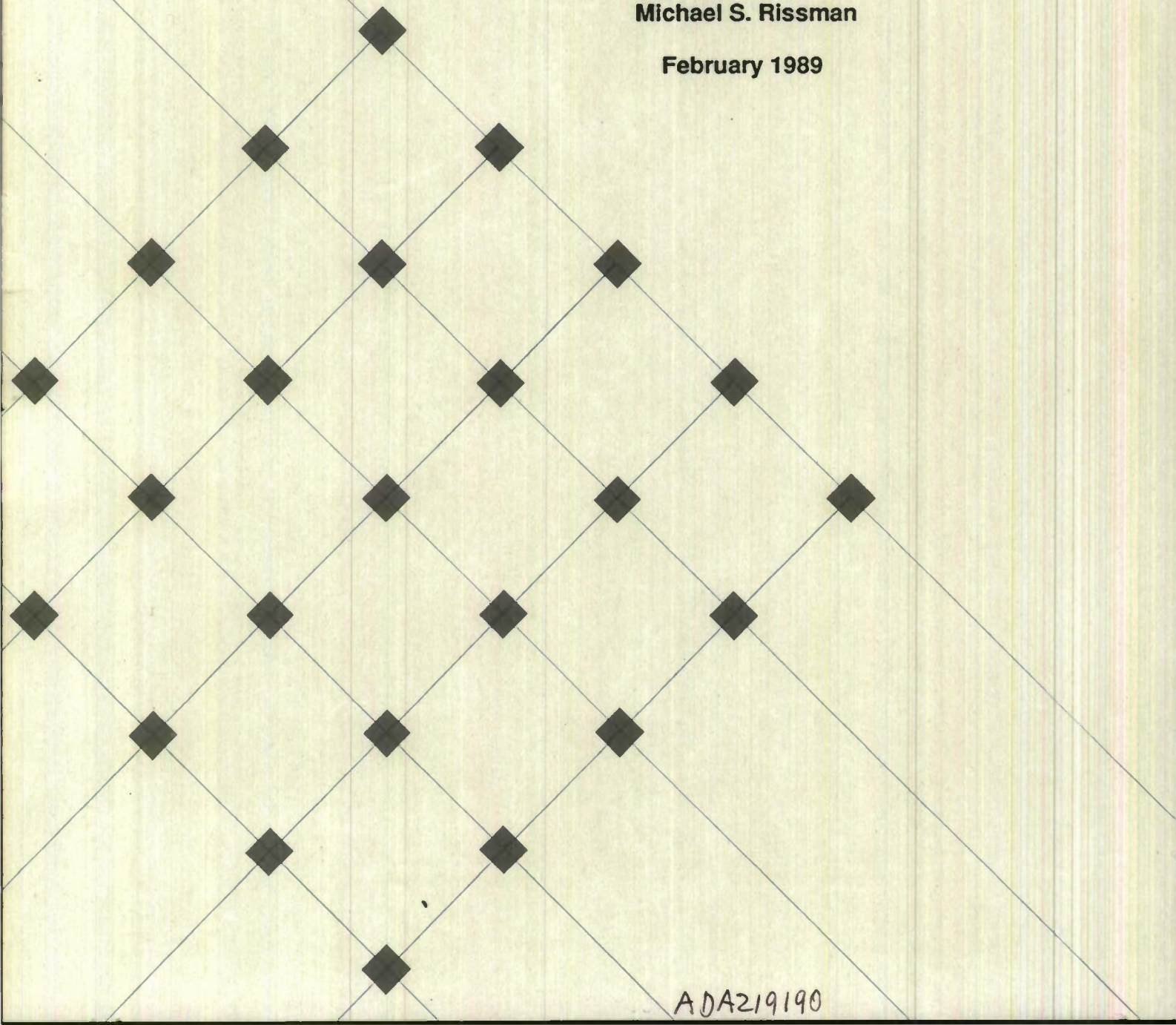
Technical Report
CMU/SEI-89-TR-5
ESD-TR-89-005

Carnegie-Mellon University
Software Engineering Institute

An Object-Oriented Solution Example: A Flight Simulator Electrical System

Kenneth J. Lee
Michael S. Rissman

February 1989



Technical Report
CMU/SEI-89-TR-5
ESD-TR-89- 005
February 1989

An Object-Oriented Solution Example: A Flight Simulator Electrical System

Kenneth J. Lee
Michael S. Rissman

Domain Specific Software Architectures Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Karl Shingler
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Table of Contents

List of Figures

Preface

This report describes an implementation of a subset of an aircraft flight simulator electrical system. It is a result of work on the Ada Simulator Validation Program (ASVP) carried out by members of the technical staff (MTS) at the Software Engineering Institute (SEI). The MTS developed a paradigm for describing and implementing flight simulator systems in general.[1] The paradigm is a model for implementing systems of objects. The objects are described in a form of design specification called an object diagram. The first report on this paradigm [3] defined package specifications, but not bodies, for an engine system. This report has been prepared to demonstrate a full implementation of a system: package specifications and bodies. The intent is to provide an example; the code is functional, but there is no assurance of robustness.

Chapter 1 presents the characteristics of an aircraft electrical system and focuses on the direct current (DC) system. Chapter 2 provides an overview of the solution by presenting the design specification and discussing the symbols used for its construction. Chapter 3 details the implementation of the objects, the focus for this report, in the DC power system example. Chapter 4 describes the implementation of the software, which allows construction of a system from a collection of objects, i.e., the connections, systems, and executives. The last part of the chapter explains the overall software architecture and the relationships between the DC system and the other systems in the simulator.

Appendix A defines the electrical concepts used in the report. Appendix B explains the software architecture diagram notation. Appendix C shows an example of a code template for object managers. Appendix D contains a complete listing of the Ada code for this example.

[1] A full description can be found in the SEI Technical Report *An OOD Paradigm for Flight Simulators, 2nd Edition*, see [3].

Abstract

This report describes an implementation of a subset of an aircraft flight simulator electrical system. It is a result of work on the Ada Simulator Validation Program (ASVP) carried out by members of the technical staff (MTS) at the Software Engineering Institute (SEI). The MTS developed a paradigm for describing and implementing flight simulator systems in general. The paradigm is a model for implementing systems of objects. The objects are described in a form of design specification called an object diagram. This report has been prepared to demonstrate a full implementation of a system: package specifications and bodies. The intent is to provide an example; the code is functional, but there is no assurance of robustness.

1. An Aircraft Electrical System

This chapter sets the context for the report by presenting the characteristics of an aircraft electrical system and focusing on the direct current (DC) system.

1.1 Overview

Electrical power is generated from aircraft engines by a drive shaft connected to generators. In one of the aircraft observed by the MTS, there are two generators attached to each aircraft engine. Each generator produces a constant 28 volts.

The electrical system is used to provide electrical power to devices in other systems, such as fuel pumps and valves in the fuel system, hydraulic pumps in the hydraulic system, and air conditioning in the environmental control system. The devices in the systems are able to function only if power is available. They, in turn, put their load, i.e., the amount of current they require, back onto the electrical system. The load is transferred back to the generators, along the electrical system buses, where a determination of possible overloading takes place. In this report, an assumption is that the generators in the electrical system and the load consumption devices in other systems are the ground points.

The electrical system on many aircraft can be divided into four systems: DC power, alternating current (AC) power, emergency power, and ground power. The AC power system consists of the generators, driven by the engines, and several AC power buses. It is connected to the other three electrical systems via relays and circuit breakers. The DC power system consists of transformer rectifier units (TRUs), which convert AC power to DC power, and several DC power buses. DC power is also connected to the emergency power system. The emergency power system is connected to both the AC and the DC systems. It has battery charging, storage, and power conversion capability between AC and DC. The ground power system connects the AC power system to ground control when the aircraft is parked. It is used for starting the engines and providing power during ground operations.

There are several components common to all the systems, e.g., buses and circuit breakers. In addition, each system has some unique components. The DC power system has TRUs, which convert

incoming AC power into DC power. The emergency power system has relays, batteries, an inverter for converting DC power into AC power, lights, and switches. The ground power system has relays, bus power control units, and power contactors.

1.2 The DC Power System

The example used in the rest of this report is a generalized view; it was synthesized from experience with several simulator projects. Some projects had systems more complex than this example, some less complex. DC power was chosen because it is intermediate in complexity among the four electrical systems. AC power is generally simpler. Emergency power is usually more complex. DC power provides enough detail to demonstrate the advantages of the paradigmatic approach.

The DC power system provides voltage to any device driven by a direct current source. DC voltage is generated from AC system voltage by six TRUs, see Figure 1. The six TRUs are powered by six AC power buses. TRUs 1, 2, and 3 are connected to the three main DC buses. The three main buses are connected together by Tie Bus 1. A tie bus maintains power to the main DC buses in the event of a failure in a TRU or an AC bus. TRUs 4, 5, and 6 are connected at Tie Bus 2. The two tie buses are connected through circuit breaker Cb_TB_1.2.

There are a total of ten DC power buses, in this example. The power output of the three main DC buses is divided between six DC power buses. Tie Bus 1 provides power to one DC power bus and Tie Bus 2 provides power to three DC power buses. These ten buses typically provide power to nearly 300 devices in the actual aircraft.

1.2.1 Sources - TRUs

A rectifier is a component for converting alternating current into direct current. In this example, the TRUs are considered the source of voltage and current for the DC power system. The system has no knowledge of any other external source of power.

The software TRUs represent components that exist in the real aircraft. The TRUs are not part of the aircraft simulator hardware; they are completely simulated in software. The TRUs have a load conversion factor (LCF) which represents the effectiveness of the conversion from AC to DC. For the purposes of this example, the load conversion factors of all the TRUs are considered to be equivalent. The TRUs also are active components which add some load to the AC buses.

1.2.2 Passive Components - Circuit Breakers

A circuit breaker is a switch that automatically interrupts an electrical circuit under abnormal conditions, usually current overloading. An assumption of this example is that passive components are ideal, i.e., there is no loss of current through a component due to the impedance of the component.

The circuit breakers model hardware components in the simulator cockpit. The instructor, at the Instructor Operator Station (IOS), has the option to trip a circuit breaker. When the instructor trips a breaker, the effect is transferred through a buffer to the simulator hardware where the breaker changes its physical position. The changed position is detected and stored in the software linkage buffer. During the update of the software system the software circuit breaker position is updated to reflect the state in the linkage buffer.

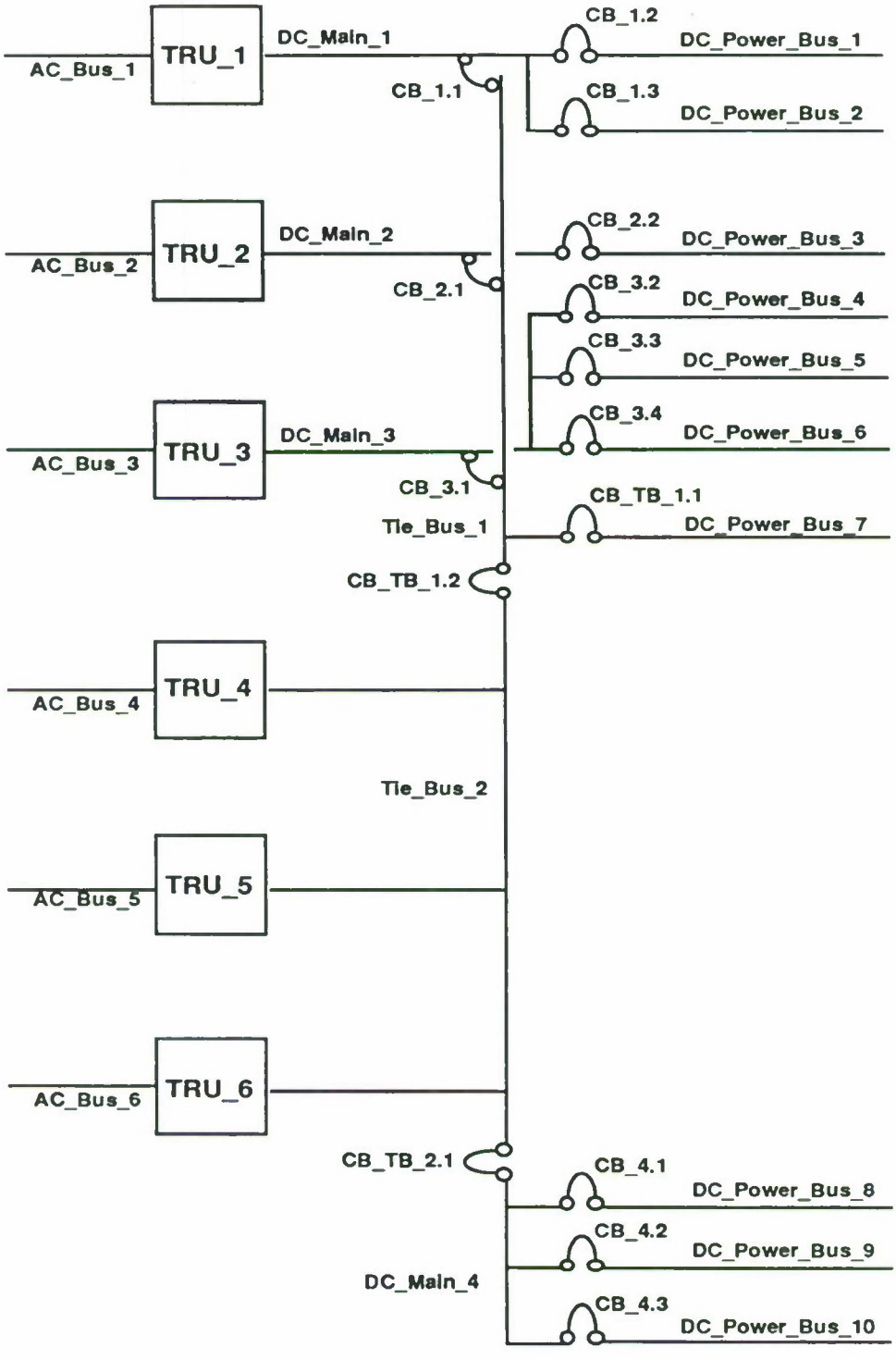


Figure 1: DC Power Circuit Diagram

Current ratings are defined for each circuit breaker. The only software generated circuit breaker fault, in this implementation, is an electrical short. In the event of an overload condition, specified by the instructor, a circuit breaker will trip, interrupting the flow of current.

1.2.3 Wires - Buses

Wires, called buses, connect all components in electrical systems. In this example, there is a bus between all other connected components.

Information flows through the components and along the buses. The software implementation of the buses implements Kirchoff's current law and voltage law (see Appendix A). Each bus has a minimum of two connections to other components.

2. An Object-Oriented Solution

This chapter provides an overview of the solution by presenting the design specification and discussing the major symbols, e.g., objects and connections, used for its construction.

2.1 Introduction

A design specification for a recurring problem is specified by an instantiation of the structural model for that problem class. A structural model is a graphic representation of a canonical solution to a problem that recurs in an application. A structural model consists of a set of symbols and rules of composition for the symbols. The model is a structural model because each symbol stands for a software template. Each instance of a symbol in the design specification is labeled to reflect the entity in the problem space it denotes. The labels are used to fill in engineering points in the templates corresponding to the symbols. Often the template for a symbol will be affected by labels on adjacent symbols and by labels of nested symbols. The design specification is implemented by effecting the mapping between symbols and templates. Implementations, built by generating instances of the templates, of a design specification to a recurring problem are then similar in structure, behavior, and functionality.

The recurring problem addressed by this report is the description of aircraft systems in terms of objects, connections, and control mechanisms. The systems may be part of one executive running on a single processor or several executives on several processors.

2.2 A Design Specification

Figure 2 is the design specification for the DC power system described in this report. It is a representation of the circuit diagram shown in Figure 1.[2] The design specification shows

- The DC power system.
- The AC power system, which is the voltage and current source for this example.
- A dummy system which represents all the load devices in other systems.

Each component, in Figure 1 (the TRUs, circuit breakers, and buses), is an object in Figure 2 and is represented as a rectangle. Arrows between the rectangles represent the connections between the objects. A double-headed arrow stands for two single-headed arrows, one arrow pointing in each direction. Two conventions are followed in this design specification:

1. There is a bus object between every other pair of connected objects.

2. Each connection has two, and only two, connection points; one on a bus object and one on another object.

All other meanings of the symbols are equivalent to the meanings defined in [3]. The templates, which map between the symbols and the implementation, are also the same. The use of the templates for this system will be discussed in Chapters 3 and 4.

This solution models the flow of information directionally. Voltage and LCF flow from voltage sources, such as generators and batteries, through passive objects to voltage sinks, such as motors and lights. Load flows from the voltage sinks back to the voltage sources. Because of this directionality, objects, such as circuit breakers and TRUs, are defined with two sides. Each side contains the information flowing through the circuit to that point. The transfer of information through an object means: obtain the stored information from the opposite side of the object. This convention maintains the proper flow through the system. Bus objects, which may be connected to any number of other objects, have as many sides as they do connection points.

The rest of this chapter briefly describes the connections and objects.

[2] The buses labeled **DC_Main_x** in Figure 1 are labeled **Main x** in Figure 2.
The buses labeled **DC_Power_Bus_x** in Figure 1 are labeled **DC x** in Figure 2.

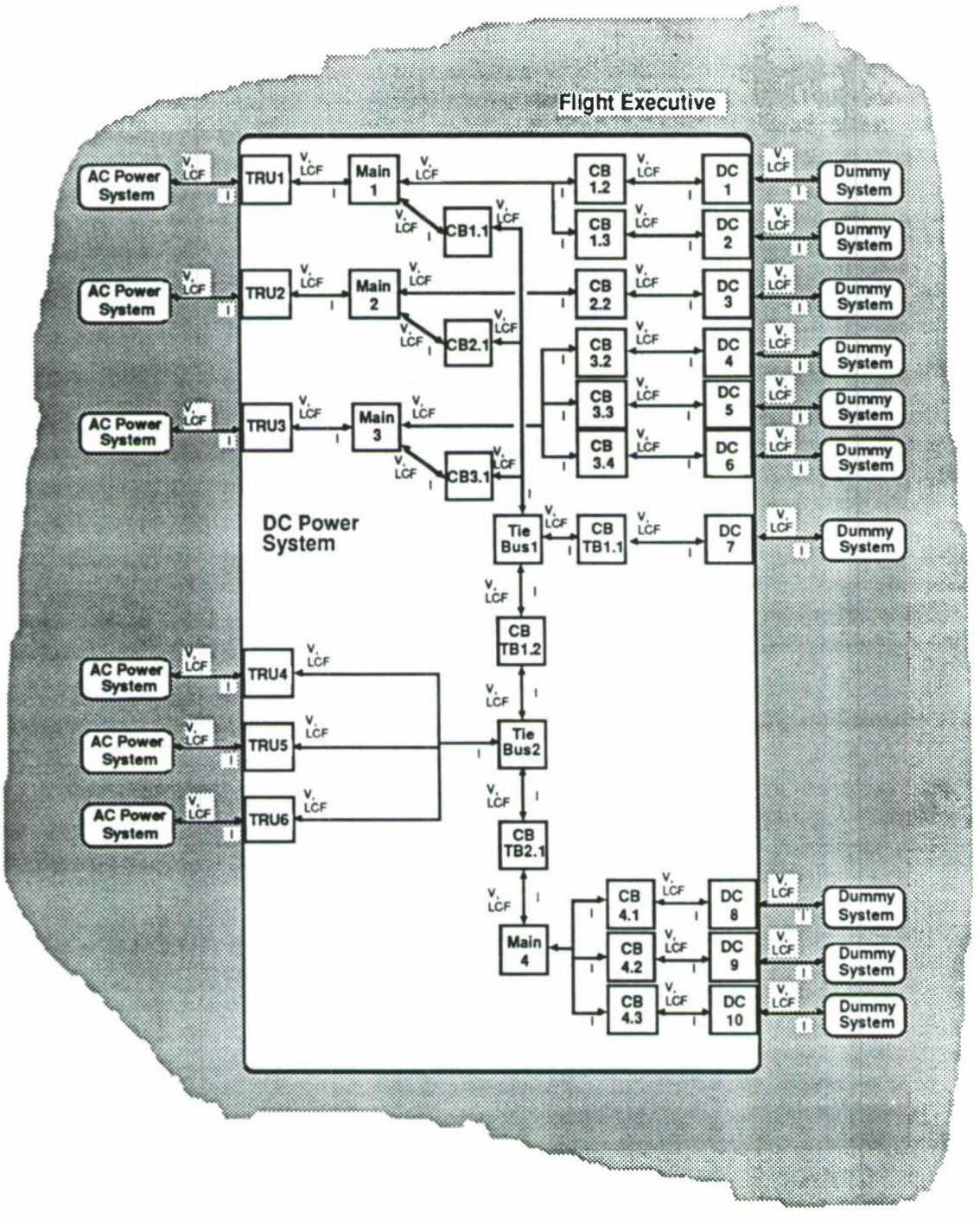


Figure 2: DC Power Design Specification

2.3 Connections

Connections are procedures which read the state of one object and write that state to another object. In Figure 2 a connection is represented by an arrow. The convention is that the labeled information is read from the object at the tail of the arrow and written to the object at the head of the arrow. For example, voltage, V, and LCF are read from AC power system bus objects and written to the TRUs in the DC power system. Current, I, is read from each TRU and written to the AC power system buses.

Connections are classified as either executive or system level connections. Executive level connections are those between systems, such as the connections between the AC power system and the DC power system. System level connections are those between objects within a system.

2.3.1 System-Level Connections

System level connections read state information from an object in the system and write the information to another object in the system. The connections are owned by the system. When the system is called to update itself, all the system level connections are gated; when complete, the system is considered to be updated.[3] Section 4.2 discusses the details of the implementation of the system connections.

Each system level connection, in Figure 2, has two connection points, one to a bus object and one to another object. Thus, even though Figure 2 may appear to have one connection between TRUs 4, 5, 6, and Tie Bus 2, the arrow represents three separate connections, one from each TRU to the tie bus.

2.3.2 Executive-Level Connections

Executive level connections read state information from an object in one system and write the information to an object in another system. In this example, the connections are owned by the flight executive. When the DC power system is updated, the flight executive connections to the system are gated, then the system is called to update itself. Section 4.4 details the implementation of the executive connections.

The DC power system has connections to AC power buses in the AC power system, which are fully implemented in this example, connections to sink devices in other systems, which are implemented as connections to objects in the dummy system, and connections to objects in the emergency power system, which are not part of this implementation. The ten DC power buses, which provide DC power to the rest of the simulator, represent all the DC buses. The connections to the dummy system objects, however, only represent ten of the hundreds of connections to actual devices that are part of a full simulator implementation. Each DC power bus would normally have dozens of connections to it, one from each device in the other systems.

---

[3] Some connections may need to be gated more than once during each system update to guarantee that the system is consistent, see Section 4.2.

2.4 Objects and Aggregates

Objects correspond to real-world entities, e.g., buses, circuit breakers, TRUs, batteries, relays, lights, switches, etc.

Some of the objects in the electrical systems are represented by physical components in the simulator cockpit as well as by software. Examples are lights, circuit breakers, and switches. A linkage buffer is available for storing the physical component state so the state is available for transfer between the cockpit and the software representation.

The other objects model physical components, but are only represented in software, e.g., TRUs and generators don't exist in the simulator cockpit.

Each object is represented by an object manager. There is a single object manager for all instances of the object.[4] The object manager defines the *attributes* of the object. The attributes are invariant characteristics defined at elaboration, e.g., a current rating of a circuit breaker.

The object manager defines the *operational state* of the object. The operational state refers to those characteristics which may change with time, e.g., the degree of charging or discharging of a battery, the setting of a switch, malfunctions, or aging effects on various components.

The object manager allows the object's *environmental effects* to be placed on the object. The environmental effects are external object states which are required by the object to determine its state.

The object manager implements the math model for the object. The math model is implementation dependent. The object managers use the math models to map the object's inputs to its outputs. The object manager produces the *outputs* available from the object. The outputs are generated by the math model, using the environmental effects placed on the object and any additional constraints imposed by the attributes and the operational state of the object. The math model may be invoked when environmental effects are placed on the object or when outputs are read from the object. This is an implementation level decision left to the system designer; it is not defined by the paradigm.

The actual instances of the objects are stored in system aggregates. An aggregate allows named access to the objects in a system; no procedure call is required to retrieve the object. The aggregate is not denoted on design specifications, but is an essential part of the implementation of a system and its objects.

The flight executive queries the DC power system's objects directly, using the DC power system aggregate, to obtain state information needed for input to other systems. The executive also queries the other systems to get the environmental effects and operational state required by the DC power system. The information is then written directly to the DC power system's objects. The DC power system uses its system aggregate to obtain state information needed for input to the other objects in the system.

---

[4] The term *manager* is used because all access to each object is administered through the interface defined by the object manager.

3. Objects

This chapter provides the details for the implementation of the objects, the focus for this report. The circuit breaker object manager will be discussed in detail. The bus and TRU object managers will be covered in less detail.

3.1 Introduction

In the previous report [3], the implementation of the connections, systems, and executives was described in some detail; however, only the package specifications of the objects were presented. The example in this report concentrates on the bodies of the objects.

A structural model, as described in Chapter 2, consists of symbols and rules for their composition. Each kind of symbol has a software code template associated with it. The template abstracts the implementation of the symbol. An instance of a symbol is created by instantiating the template with the appropriate substitutions for the placeholders. Placeholders are replaceable snippets of code in templates which stand for Ada types, names, and values. There are two types of placeholders:

1. The first is of the form *{Object}* or *{Attribute}*, i.e., a phrase enclosed in brackets. The entire phrase (including the brackets) must be replaced. For example, *{Object}_Object_Manager* becomes *TRU_Object_Manager* for all instances of *{Object}_Object_Manager* in a file when *{Object}* is replaced by *"TRU"*.

2. The second form is the double question mark, *??*. This form means that some special action must be taken by the user. For example, this form is used for those places in the code that the user needs to supply a function body, supply test cases, or remove some lines from the template, e.g., the instructions at the beginning of each template.

Appendix C contains a code template for object managers. The template allows the production of object managers that are similar in appearance and documentation. Much of the documentation for the object manager is contained in the template and is filled in at the time the placeholders are replaced. Some specific comments must be added to customize an instance of the template for a particular object manager. Also, the bodies of the procedures, which provide the mapping of the inputs to the outputs, must be supplied, since each object manager implements the math model for a different kind of object. A template also contains a test routine which allows some unit testing of the object

manager. The test routine is filled in during replacement of the placeholders and is ready to use once the object manager is compiled.

3.2 Global Types

The object managers, produced from the template, are only dependent on global types. For the DC power system, the global types package is called **Electrical_Units** (Figure 3).

The package **Electrical_Units** provides the voltage, current, and LCF definitions. It also declares common constant values for those types. Voltage is an enumerated type. Current and LCF are real values.

The aggregate record, Power_Info, contains components for voltage, current, and LCF. Power_Info is used in most object processing routines for accessing the electrical values of an object.

```
package Electrical_Units is

    type Voltage is (Floating_Voltage, Zero_Voltage, Available_Voltage);
    No_Voltage : constant Voltage := Zero_Voltage;

-- Devices like relays need to know if
-- voltage is available without concern for the level.
--
    Energizing_Voltage : constant Voltage := Available_Voltage;

    type Current is new Float;
    No_Current : constant Current := 0.0;

    type Load_Conversion_Factor is new Float;
    No_Load_Conversion : constant Load_Conversion_Factor := 0.0;

-- Needed when device shorts out when current passes the wrong way.
--
Max_Load_Conversion : constant Load_Conversion_Factor := 10_000.0;

-- Permits a function to return all three values.
--
    type Power_Info is
        record
            V : Voltage := Floating_Voltage;
            I : Current := No_Current;
            Lcf : Load_Conversion_Factor := No_Load_Conversion;
        end record ;
end Electrical_Units;
```

Figure 3: Package Electrical_Units

3.3 The Circuit Breaker Object Manager

Figure 4 shows a package specification for the **Cb_Object_Manager**. Type Cb is implemented as a private type. A new instance of the type Cb is returned by the function New_Cb. The Cb's attribute, Rating, and operational state, Position, definitions are shown as well. The definitions are part of the specification and may be accessed by the system aggregates and connection routines. Operations are defined for providing the external state information, voltage, LCF, and current, to a circuit breaker and for reading the state, the Power Info, of the circuit breaker. Operations are also defined for setting and reading the value of the operational state, Position. All the operations take a named instance of the Cb as a parameter. In the private part of the package specification, the breaker's private type is declared as an access pointer to a data type which will be the actual representation of the object. The data type is an incomplete type, the details of which are delayed until the package body.

```
with Electrical_Units;
package Cb_Object_Manager is

    package Eu renames Electrical_Units;

    type Cb is private;
    type Cb_Position is (Open, Closed);
    type Cb_Rating is new Float;

    type Cb_Side_Names is (Side_1, Side_2):

    function New_Cb (Position : in Cb_Position;
          Rating : in Cb_Rating)
          return Cb;

    procedure Give_Voltage_Lcf_To (A_Cb : in Cb;
             A_Cb_Side : in Cb_Side_Names;
             Volts : in Eu.Voltage;
             Load_Conversion : in Eu.Load_Conversion_Factor);

    procedure Give_Current_To (A_Cb : in Cb;
                               A_Cb_Side : in Cb_Side_Names;
                               Load : in Eu.Current);

    procedure Give_Power_Info_To (A_Cb : in Cb;
          A_Cb_Side : in Cb_Side_Names;
          External_Power_Info : in Eu.Power_Info);

    function Get_Power_Info_From (A_Cb : in Cb;
         A_Cb_Side : in Cb_Side_Names)
         return Eu.Power_Info;

    procedure Give_Position_To (A_Cb : in Cb;
         Position : in Cb_Position);

    function Get_Position_From (A_Cb : in Cb) return Cb_Position;

pragma Inline (New_Cb,
          Give_Voltage_Lcf_To,
          Give_Current_To,
          Give_Power_Info_To,
          Get_Power_Info_From,
          Give_Position_To,
          Get_Position_From);

private
    type Cb_Representation;              -- incomplete type, defined in package body
    type Cb is access Cb_Representation; -- pointer to an Cb representation
end Cb_Object_Manager;
```

Figure 4: Cb_Object_Manager Package Specification

Figure 5 contains the package body for the **Cb_Object_Manager.** The record representing the circuit breaker object is called Cb_Representation. The record contains components for the attribute, Rating, the operational state, Position, and the external environmental effects, Points. The Points component is an array of Power_Info records, indexed by the side names of the circuit breaker.

The function Opposite_Side returns the Power Info record for the opposite side of a circuit breaker. Recall from Chapter 2, this example models the flow of information directionally. Because of this directionality, objects, such as circuit breakers, are defined with two sides. Each side contains the information flowing through the circuit to that point. The transfer of information through a object means: obtain the stored information from the opposite side of the object. This convention maintains the proper flow through the system.

The function New_Cb creates a new instance of a circuit breaker object and returns an access pointer to the private type. In addition, the function sets values for the circuit breaker's current rating and physical position. The new instance is named and stored in the system aggregate, see Chapter 4.

The implementation of the subprogram bodies for reading and writing external effects is not specified within the paradigm. The math models will be specific to each kind of object. The behavior of the subprograms will be that they update the object's state on a write operation or on a read operation. The implementation of function Get_Power_Info_From calculates the circuit breaker's state when the output is read, i.e., when the circuit breaker is closed the Power Info record from the opposite side of the circuit breaker is returned, otherwise a zeroed Power Info record is returned. Because this version of the **Cb_Object_Manager** determines state on a read operation, the write operations, Give_Position_To, Give_Voltage_Lcf_To, and Give_Current_To, simply write values to the Cb_Representation record.

A more complete solution for an object manager would include symmetrical "Give_" and "Get_" procedures. For example, **Cb_Object_Manager** has a Give_Voltage_Lcf_To procedure but no Get_Voltage_Lcf_From procedure. The Get_Power_Info_From procedure is used in all cases when any part of the Power_Info record needs to be accessed.

Time effects would be placed on an object by the flight executive in the same way as other external environmental effects. For example, if a battery had to charge (or discharge) over time, during each system update the battery object would receive a value of the current time. The battery state would then be a reflection of the rate of charging (or discharging) over time.

```
package body Cb_Object_Manager is

    type Point_Representation is array (Cb_Side_Names) of Eu.Power_Info;

    type Cb_Representation is
        record
            Points: Point_Representation;
            Position: Cb_Position:= Closed;
            Rating: Cb_Rating:= 50.0;
        end record;

    function Opposite_Side (This_Side: in Cb_Side_Names) return Cb_Side_Names is
    --/*****************************************************************
        The_Side: Cb_Side_Names:= Side_1; -- one of the sides
    begin
        -- select opposite side based on what this side is.
        if This_Side = Side_1 then
            The_Side:= Side_2; -- the other side
        end if;
        RETURN The_Side;
    end Opposite_Side;

    function New_Cb (Position : in Cb_Position;
                     Rating : in Cb_Rating)  return Cb is
    --/*****************************************************************
        The_New_Object: Cb:= new Cb_Representation;
    begin
        The_New_Object.Position:= Position;
        The_New_Object.Rating:= Rating;
        RETURN The_New_Object;
    end New_Cb;

    procedure Give_Voltage_Lcf_To (A_Cb : in Cb;
                                   A_Cb_Side : in Cb_Side_Names;
                                   Volts : in Eu.Voltage;
            Load_Conversion : in Eu.Load_Conversion_Factor) is
    --/*****************************************************************
    begin
        A_Cb.Points (A_Cb_Side).V:= Volts;
        A_Cb.Points (A_Cb_Side).Lcf:= Load_Conversion;
    end Give_Voltage_Lcf_To;

    procedure Give_Current_To (A_Cb : in Cb;
                               A_Cb_Side : in Cb_Side_Names;
                               Load : in Eu.Current) is
    --/*****************************************************************
    begin
        A_Cb.Points (A_Cb_Side).I:= Load;
    end Give_Current_To;
```

Figure 5: CB_Object_Manager Package Body

```
    procedure Give_Power_Info_To (A_Cb : In Cb;
                                  A_Cb_Side : In Cb_Side_Names;
                                  External_Power_Info : In Eu.Power_Info) Is
    --/*****************************************************************
    begin
        A_Cb.Points (A_Cb_Side):= External_Power_Info;
    end Give_Power_Info_To;

    function Get_Power_Info_From (A_Cb : In Cb;
                                  A_Cb_Side : In Cb_Side_Names)
                                  return Eu.Power_Info Is
    --/*****************************************************************
        The_Power_Info: Eu.Power_Info;
    begin
        If A_Cb.Position = Closed then
            The_Power_Info:= A_Cb.Points (Opposite_Side (A_Cb_Side));
        end If;
        RETURN The_Power_Info;
    end Get_Power_Info_From;

    procedure Give_Position_To (A_Cb : In Cb;
         Position : In Cb_Position);
    --/*****************************************************************
    begin
        A_Cb.Position:= Position;
    end Give_Position_To;

    function Get_Position_From (A_Cb : In Cb)
         return Cb_Position;
    --/*****************************************************************
    begin
        RETURN A_Cb.Position;
    end Get_Position_From;
end Cb_Object_Manager;
```

Figure 5: CB_Object_Manager package body – concluded

An alternative implementation of the package body would determine the output state during a write operation. For this math model, a new component, Output_State, would be added to the Cb_Representation record, see Figure 6.

```
type Cb_Representation is
   record
      Points: Point_Representation;
       Position   : Cb_Position:= Closed;
       Rating     : Cb_Rating:= 50.0;
     Output_State: Point_Representation;
   end record;
```

Figure 6: Alternative Cb_Representation

The procedures Give_Power_Info_To, Give_Position_To, Give_Voltage_Lcf_To, and Give_Current_To write the external environmental effects. In addition to writing the effects, these procedures must now determine the output state and write the Output_State component in the Cb_Representation record. Implementations of Give_Power_Info_To and Give_Position_To are shown in Figure 7 and Figure 8, respectively.

```
procedure Give_Power_Info_To (A_Cb: In Cb;
                              A_Cb_Side: In Cb_Side_Names;
                              External_Power_Info: In Eu.Power_Info) is
    The_Power_Info: Power_Info;

    begin
        A_Cb.Points (A_Cb_Side):= External_Power_Info;

        If A_Cb.Position = Closed then
            A_Cb.Output_State (Opposite_Side(A_Cb_Side)):= External_Power_Info;
        else
            A_Cb.Output_State (Opposite_Side(A_Cb_Side)):= The_Power_Info;
        end If;
end Give_Power_Info_To;
```

Figure 7: Alternative Give_Power_Info_To Procedure

```
procedure Give_Position_To (A_Cb: In Cb;
                            Position: In Cb_Position) is
    The_Power_Info: Power_Info;

    begin
        A_Cb.Position:= Position;

        If A_Cb.Position = Open then
            for A_Side in Cb_Side_Names loop
                A_Cb.Output_State (A_Side):= The_Power_Info;
            end loop;
        end if;
end Give_Position_To;
```

Figure 8: Alternative Give_Position_To Procedure

For this alternative implementation, the function Get_Power_Info_From must return the Output_State, see Figure 9. Thus, the math model may be invoked when outputs are read from the object, as in Figure 5, or when environmental effects are placed on the object, as in Figure 6, Figure 7, and Figure 8. This is an implementation level decision left to the system designer; it is not defined by the paradigm.

```
function Get_Power_Info_From (A_Cb: In Cb;
                              A_Cb_Side: In Cb_Side_Names)
                              return Power_Info is

    begin
        RETURN A_Cb.Output_State (A_Cb_Side);
end Get_Power_Info_From;
```

Figure 9: Alternative Get_Power_From Function

3.4 Other Objects

This section will briefly describe the object managers for the bus objects and TRU objects. The structure of the object managers is the same as the **Cb_Object_Manager.** The behavior of the object managers is the same, i.e., object states are updated on read operations. The functionality reflects the math models of the particular objects.

3.4.1 Bus_Object_Manager

Electrical buses connect all other components. The details of how voltages, LCFs, and loads are propagated are contained in the subprogram bodies of the **Bus_Object_Manager.** The Bus_Representation record, see Appendix D, Section D.9, saves the input state of each connection, the output state of each connection, a dirty bit for voltage, and a dirty bit for current. The subprogram bodies calculate the output states during read operations. During a write operation, a dirty bit is set to true only if the value of voltage or current being written is different than the current output state of the bus.[5] During a read operation, output states are recalculated only if a dirty bit is set. If recalculation is necessary, the output states for all connections are determined and all dirty bits are set to false.

3.4.2 TRU_Object_Manager

The TRUs are directional components. They transmit voltage from the AC side to the DC side and transmit load from the DC side to the AC side. The TRU_Representation record, see Appendix D, Section D.7, contains the attributes of the TRU, Tru_LCF and Tru_Load, and the input states of both sides. There is no transformation of voltage or current within a TRU in this implementation. Thus, the DC side of the TRU provides the voltage that arrives from the AC side of the TRU, to the rest of the DC power system. Also, the AC side of the TRU provides the load, which is the sum of the load to the DC side of the TRU and the TRU's own contribution, to the AC power system.

[5] There is no dirty bit for LCF. A changed value of LCF affects current. So, if a new LCF value is written to a bus the dirty bit for current is set.

4. Connections, Systems, and Executives

This chapter describes the implementation of the software that allows construction of a system from a collection of objects, i.e., the connections, systems, and executives. The last part of the chapter explains the software architecture and the relationships between the DC system and the other systems in the simulator.

4.1 DC Power System Aggregate

The DC power system aggregate package, see Appendix D, Section D.10, defines the instances of all the objects within the system. The objects are named and the names are used to reference an array of pointers to the instances. Figure 10 shows part of the aggregate package; the definition of a few circuit breakers.

```
type Cb_Names is (
    -- CB's between TRUs and dc_tie_bus
    Cb_1_1, Cb_2_1, Cb_3_1, . . .);

    -- define a table in which the objects are instantiated and can be referenced by the name.
    Named_Cbs : constant array (Cb_Names) of Cb_Object_Manager.Cb := (
        Cb_1_1 => Cb_Object_Manager.New_Cb
            Position => Cb_Object_Manager.Closed, Rating => 50.0),
        Cb_2_1 => Cb_Object_Manager.New_Cb
            (Position => Cb_Object_Manager.Closed, Rating => 50.0),
        Cb_3_1 => Cb_Object_Manager.New_Cb
            (Position => Cb_Object_Manager.Closed, Rating => 50.0), . . .);
```

Figure 10: DC Power System Aggregate Package Fragment

The New_{object} function in each object manager is used to create the instances.[6] The names are used by connection packages to access the objects directly using the constant arrays. The aggregate

[6] The use of "{...}" within subprogram names, type names, or text refers to a general case of the item. For example, **New_{object}** is a general form representing all instances of the procedure name, e.g., **New_Cb, New_Tru,** and **New_Bus.** See the template in Appendix C for a more complete example.

package is referenced by both the DC power system connection procedures and the connection procedures in the flight executive connection package.

4.2 DC Power System and Its Connections

The DC power system owns all connections between the TRUs, circuit breakers, and the buses within its boundary. The connections move information between bus objects and other objects. The processing of the connections results in the movement of the voltages, currents, and LCFs through the system.

The data structure defined for the connections is Dc_Power_System_Connections, shown in Figure 11. The structure maps enumerated connection names to contact points on objects. Each connection has two contact points.

```
The_Dc_Power_System_Connections : constant Dc_Power_System_Connections := (

    Connection_7 => (
        1 => (Element => Global_Types.A_Tru,
              Tru_Element => Dcpa.Tru_1,
              Tru_Side => Tru_Object_Manager.Dc_Side),
        2 => (Element => Global_Types.A_Bus,
              bus_Element => Dcpa.Dc_Main_1,
              bus_Side => 1)),

    Connection_8 => (
        1 => (Element => Global_Types.A_Tru,
              Tru_Element => Dcpa.Tru_2,
              Tru_Side => Tru_Object_Manager.Dc_Side),
        2 => (Element => Global_Types.A_Bus,
              bus_Element => Dcpa.Dc_Main_2,
              bus_Side => 1)),

    Connection_9 => (
        1 => (Element => Global_Types.A_Tru,
              Tru_Element => Dcpa.Tru_3,
              Tru_Side => Tru_Object_Manager.Dc_Side),
        2 => (Element => Global_Types.A_Bus,
              bus_Element => Dcpa.Dc_Main_3,
              bus_Side => 1)),

    .
    .
    .
);
```

Figure 11: Connection Data Structure

For voltage and LCF, the connections are processed in order from the voltage sources to the voltage sinks. The enumerated connection names are defined in this order. For load, the connections are processed in reverse order. The DC power system has two main tie buses. The tie buses maintain

power to all the DC buses in the event of a failure in a TRU or an AC bus. Voltage and current move in both directions around the tie buses. As a result, the connections to the tie buses need to be processed more often than the other connections. Procedure Update_Dc_Power_System, in Figure 12, shows the order of processing of the system level connections and the extra processing required for the tie bus connections.

```
procedure Update_Dc_Power_System is
—|*****************************************************************
—| Description:
—| Upon completion of the execution of this procedure, all objects within DC power will be updated and
—| in a consistent state.  At this level of DC power the connections between objects are all that need to
—| be processed.
—|*****************************************************************
begin
— Process all connections which are internal to Dc_Power_System for voltage and lcf
—
    for A_Connection in Dc_Power_System_Connection_Names'First..
                Dc_Power_System_Connection_Names'Last loop
        Process_Voltage_Lcf_For_Connection (A_Connection);
    end loop ;

    for A_Connection in reverse The_Tie_Bus_1_Connections'First..
                The_Tie_Bus_1_Connections'Last loop
        Process_Tie_Bus_Voltage_Lcf (A_Connection);
    end loop ;

    for A_Connection in reverse The_Tie_Bus_2_Connections'First..
                The_Tie_Bus_2_Connections'Last loop
        Process_Tie_Bus_Voltage_Lcf (A_Connection);
    end loop ;

— Process all connections which are internal to Dc_Power_System for load
—
    for A_Connection in reverse Dc_Power_System_Connection_Names'First..
                Dc_Power_System_Connection_Names'Last loop
        Process_Load_For_Connection (A_Connection);
    end loop ;

    for A_Connection in The_Tie_Bus_1_Connections'First..The_Tie_Bus_1_Connections'Last loop
        Process_Tie_Bus_Load (A_Connection);
    end loop ;

    for A_Connection in The_Tie_Bus_2_Connections'First..The_Tie_Bus_2_Connections'Last loop
        Process_Tie_Bus_Load (A_Connection);
    end loop ;
end Update_Dc_Power_System;
```

Figure 12: System Update Routine

Each connection is processed in turn. The first step is to gather the information from one side of the connection, convert the information if necessary, then write the information to the other side of the connection. The system aggregate is used to access the object's state information. Figure 13 shows part of a typical connection processing routine. Converting the information is unnecessary for this routine.

```
procedure Process_Voltage_Lcf_For_Connection
                    (This_Connection: Dc_Power_System_Connection_Names) is
—|*****************************************************************
—| Description:
—|       This procedure processes the voltage and lcf state variables for the specified connection.
—|
—| Parameter Description:
—|       This connection is the connection to be updated
—|*****************************************************************
    The_Power_Info : Electrical_Units.Power_Info;

 begin
    declare
        The_Connection : A_Dc_Power_System_Connection
            renames The_Dc_Power_System_Connections (This_Connection);
    begin
    — handle a connection; obtain power information from dc power object
        case The_Connection (1).Element is
            when Global_Types.A_Cb =>
                The_Power_Info := Cb_Object_Manager.Get_Power_Info_From (
                    A_Cb => Dcpa.Named_Cbs (The_Connection (1).Cb_Element),
                    A_Cb_Side => The_Connection (1).Cb_Side);
    .
    .
    .
        end case;
    — restore new Voltage and Lcf states to an object
        case The_Connection (2).Element is
            when Global_Types.A_Wire =>
                Wire_Object_Manager.Give_Voltage_Lcf_To (
                    A_Wire => Dcpa.Named_Wires (The_Connection (2).Wire_Element),
                    A_Wire_Side => The_Connection (2).Wire_Side,
                    Volts => The_Power_Info.V,
                    Load_Conversion => The_Power_Info.Lcf);
        end case;
    end;
end Process_Voltage_Lcf_For_Connection;
```

Figure 13: Connection Processing Routine

4.3 DC Power System Software Architecture

Figure 14 shows the software architecture from the perspective of the DC power system. Objects in a system are created and named by the **{system_name}_System_Aggregate** package. Objects are managed by **{object_name}_Object_Manager** (OM) packages. The system level connections are in the body of the system package, rather than in a separate connection package. The separate package is drawn for notational simplicity. See Appendix B for a description of the icons used. The arrows represent Ada (*withing*) dependencies. The shaded portion of each icon represents the package body, the white portion represents the package specification. Note that all dependencies originate within the package bodies. This reduces the need for widespread recompilation in the event of a change.

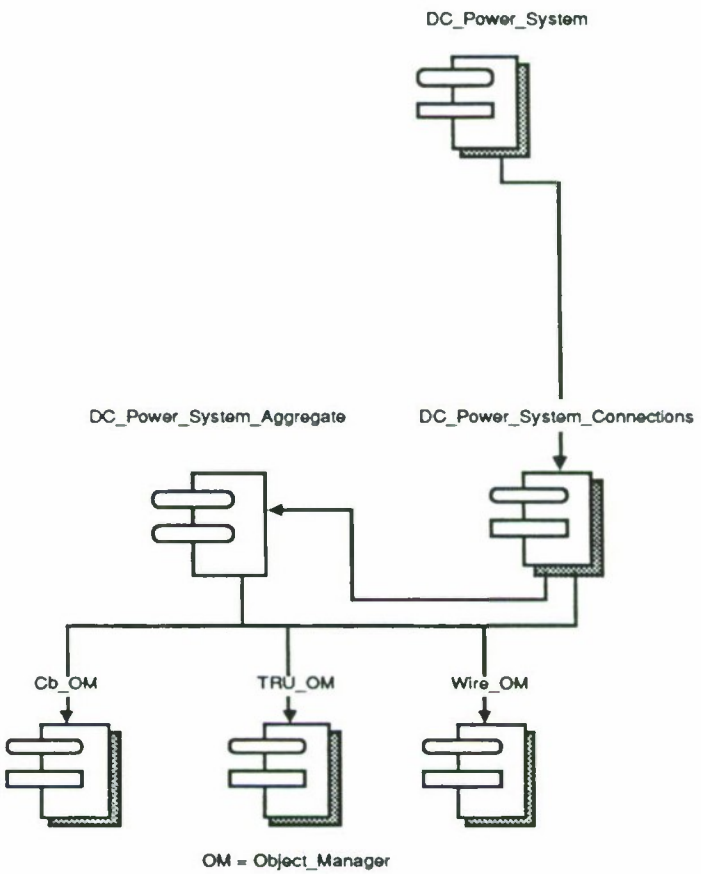


Figure 14: System-Level Software Architecture

4.4 Flight Executive and Its Connections

The flight executive level consists of two packages. The first is the **Flight_Executive** package. The body of the **Flight_Executive** package contains a tabular schedule of systems to update. The names of the systems are declared in the package **Flight_System_Names**, the sole purpose of which is to enumerate the names. Figure 15 shows a typical schedule table. In this schedule table, the AC power system is updated during frame 2, the DC power system during frame 4, and the dummy system during frame 6.

```
-- Define the allocation of processing relative to frame execution
Its_Time_To_Do : constant array (Global_Types.Execution_Sequence) of Active_In_Frame := (

    Global_Types.Frame_1_Modules_Are_Executed =>
        (Fsn.Engine_1 => (True), others => (False)),

    Global_Types.Frame_2_Modules_Are_Executed =>
        (Fsn.Ac_Power => (True), others => (False)),

    Global_Types.Frame_3_Modules_Are_Executed =>
        (Fsn.Engine_2 => (True), others => (False)),

    Global_Types.Frame_4_Modules_Are_Executed =>
        (Fsn.Dc_Power => (True), others => (False)),

    Global_Types.Frame_5_Modules_Are_Executed =>
        (Fsn.Engine_3 => (True), others => (False)),

    Global_Types.Frame_6_Modules_Are_Executed =>
        (Fsn.Dummy => (True), others => (False)),

    Global_Types.Frame_7_Modules_Are_Executed =>
        (Fsn.Engine_4 => (True), others => (False)),

    Global_Types.Frame_8_Modules_Are_Executed =>
        (others => (False)));
```

Figure 15: Executive Schedule Table

The procedure Update_Flight_Executive updates systems based on the schedule table, see Figure 16. Updating a system means to gate all executive level connections to the system and then call the system's Update_{system}_System procedure. For the DC power system, the state of the circuit breakers in the software must be updated to reflect the state in the simulator hardware on each cycle. This update is handled by the procedure Process_Cb_Linkages.

```
procedure Update_Flight_Executive
    (Frame : In Global_Types.Execution_Sequence) Is
    --|*************************************************************
    --| Description:
    --|  flight systems executive. Processes connections and updates for each system atomically.
    --|
    --| Parameter Description:
    --|  frame is the current frame
    --|*************************************************************

begin
    for A_System In Fsn.Name_Of_A_Flight_System loop
        If Its_Time_To_Do (Frame) (A_System) then

            case A_System Is
                when Fsn.Dc_Power =>
                    -- update CB linkages -- from simulator hardware
                    Flight_Executive_Connections.Process_Cb_Linkages;
                    -- Process Connections
                    Flight_Executive_Connections.Process_External_Connections_To_Dc_Power;
                    -- Update system
                    Dc_Power_System.Update_Dc_Power_System;
                when others =>
                    null;
            end case;
        end If;
    end loop;
end Update_Flight_Executive;
```

Figure 16: Update_Flight_Executive Procedure

4.5 Flight Executive Software Architecture

Figure 17 shows the executive level software architecture. In this case, we assume an executive level called Flight_Executive. Each system is represented by a package called **{system_name}_System**. The specification of each system package exports a single procedure, Update_{system_name}_System, which is called by the flight executive to update the system.

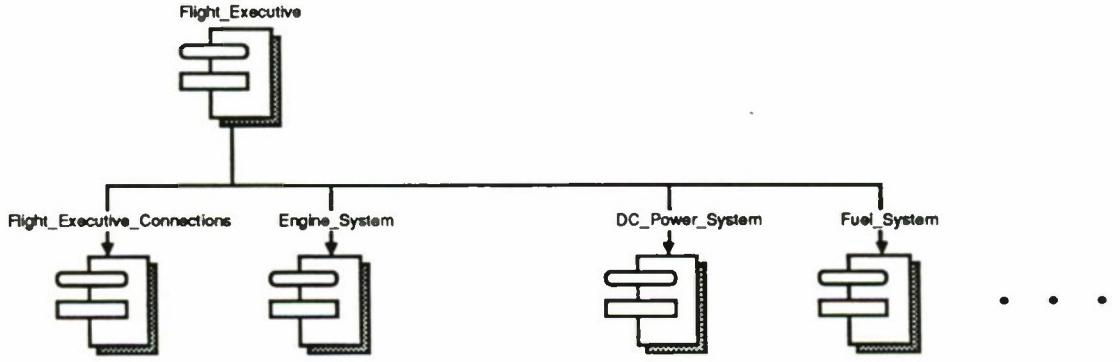


Figure 17: Executive-Level Software Architecture

The flight executive handles all connections between the DC power system and the other systems in the simulator, e.g., the engine system, the AC power system, the fuel system, and so on. The executive level connections are managed by an **{executive_name}_Connections** package, in this case, **Flight_Executive_Connections**. The architecture from the perspective of the connection package is shown in Figure 18.

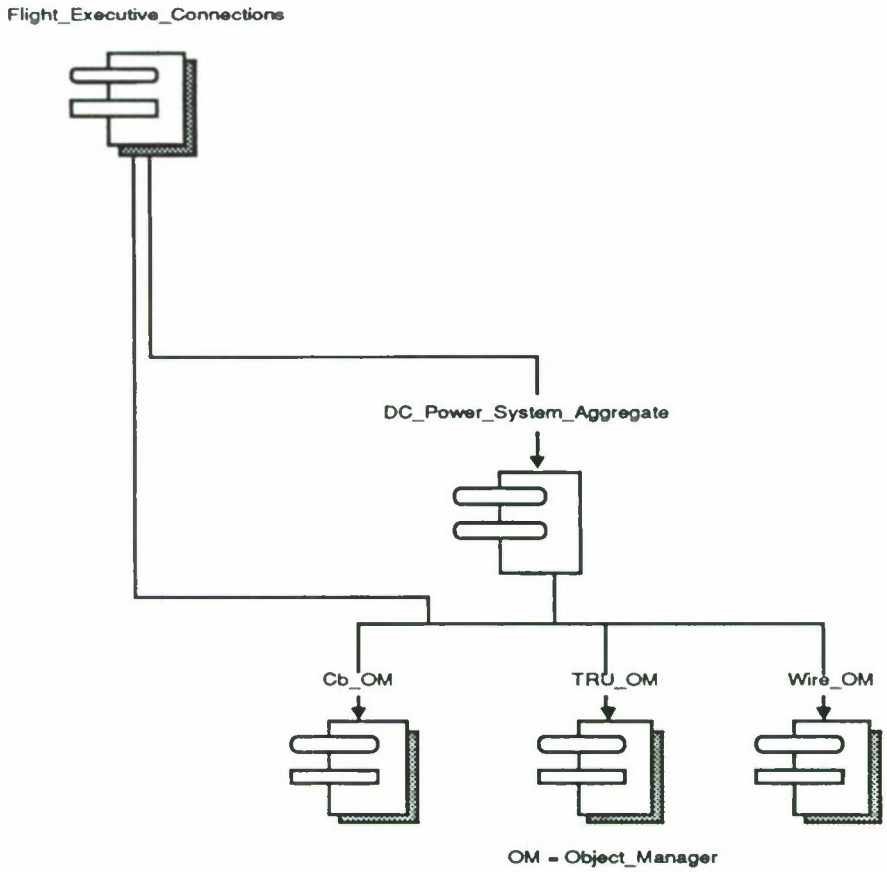


Figure 18: Executive-Connection-Level Software Architecture

The body of the connection package is a series of separate procedures, one for each system under the control of the executive. Each separate procedure is responsible for gating all the executive level connections to a system.

The data structure in the flight executive connection package, see Appendix D, Section D.19, will resemble the connection data structure for the DC power system in Figure 11. The enumerated connection names map to contact points on objects.

The order of executive level connection processing may be important in some implementations. For this example, the order is not important. As with the system connections, each connection is processed in turn. The first step is to gather the information from an object in one system, using the system aggregate for that system, convert the information, if necessary, then write the information to an object in a second system using the system aggregate for the second system.

4.6 Overall Software Architecture

The overall software architecture is shown in Figure 19. The executive level consists of the **Flight_Executive** package and the **Flight_Executive_Connections** package. The system level consists of

- **{system_name}_System** packages
- **{system_name}_System_Connections** packages
- **{system_name}_System_Aggregate** packages

The complete system level architecture of the DC power system is shown. The architecture of the other systems, e.g., the fuel system and the engine system, would be similar.

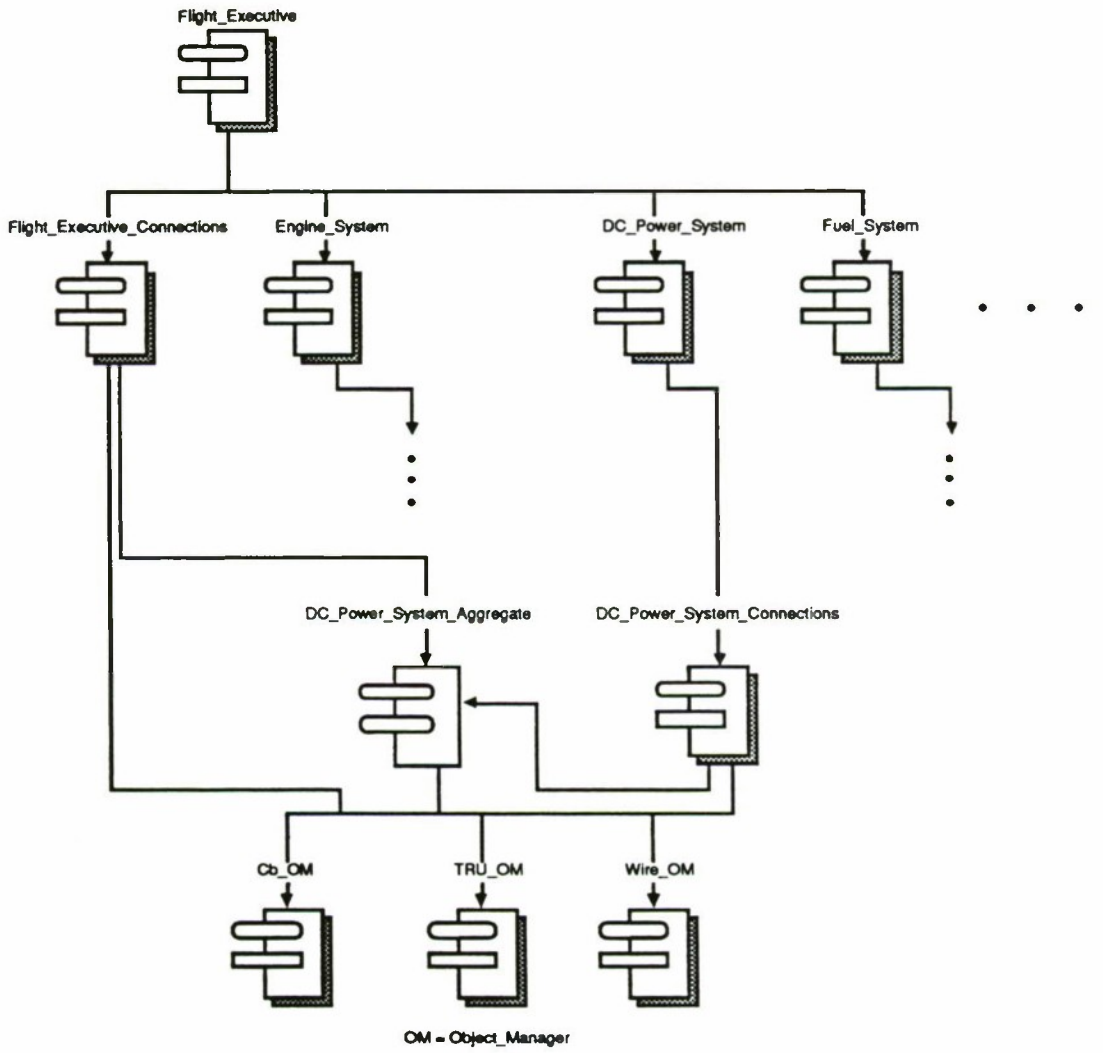


Figure 19: Overall Software Architecture

Each connection package is "nested" within the corresponding system or executive packages. Each connection within the connection packages is distinct, embodied within a separate procedure.

There is one object manager package per kind of object. The object manager packages have no dependency on other compilation units, except perhaps global types packages, e.g., **Electrical_Units**. The instances of the objects are created in the system aggregate package using the facilities of the object manager packages.

Finally, as is evident from Figure 19, compilation dependencies are limited to a system. The scope of a system does not propagate beyond the **Flight_Executive_Connections** package body. Thus a change, or complete replacement, of a system only affects the system level packages and the body of the executive connections package. No other systems are affected. The effects of changes will be transferred correctly by the executive connections during system update.

References

[1] Booch, Grady.
Software Components with Ada.
The Benjamin Cummings Publishing Company, Inc., Menlo Park, CA, 1987.

[2] Booch, Grady.
Software Engineering with Ada.
The Benjamin Cummings Publishing Company, Inc., Menlo Park, CA, 1987.

[3] Lee, K.J., Rissman, M.S., D'Ippolito, R., Plinta, C., Van Scoy, R.
An OOD Paradigm for Flight Simulators, 2nd Edition.
Technical Report, CMU/SEI-88-TR-30, Software Engineering Institute, Pittsburgh, PA, 1988.

Appendix A. Electrical Concepts

The relevant concepts for this report are:

Current (I) The transfer of electric charge. In direct current (DC) circuits, this transfer is unidirectional; in alternating current (AC) circuits, the transfer alternates on a periodic basis according to the frequency of the impressed voltage. The units for current are amperes.

Kirchoff's Current Law

The algebraic sum of the currents into a node is zero. For this report, currents into a node are defined as positive and currents out of a node as negative.

Kirchoff's Voltage Law

The algebraic sum of the element voltages in a closed circuit is zero. For this report, voltage increases (from sources such as generators and batteries) are defined as positive and voltage decreases (from sinks such as motors or lights) as negative.

Load The power consumption of a device. It is the device's requirement for current at its rated operating voltage.

Load Conversion Factor (LCF)

An indication of the relative ability of a TRU to share a common load.

Ohm's Law The current through a passive circuit element is equal to the impressed voltage divided by the element impedance. In DC circuits, the impedance is simply the resistance.

Voltage (V) A measure of electromotive force, or electrical potential; the ability to do electrical work. The units for voltage are Volts.

Appendix B. Software Architecture Notation

The notation used to describe the software architecture in this report is a modified form of the notation expounded by Grady Booch in his book on software engineering with Ada [2] and his book on reusable software components with Ada [1]. The notation used is true to the intent of Booch's notation. The variations are:

- The use of reduced package and subprogram icons inside larger icons rather than the object (or blob) icon.
- The use of object dependency arrows, more subtly, to distinguish different types of dependencies.
- Internal details of any reusable subsystem, package, subprogram or task are not shown.

One final note about the notation: The figures do not need to show all the fine-grained detail of a package or subprogram. When the code of a package (or subprogram) is compared to a figure associated with that package (or subprogram) there may be nested procedures or packages not shown on a particular picture, or it may depend on a package not explicitly shown in the figure. The guidelines for these cases are:

- Utility packages or services are not shown (this includes things like text_io, reusable data structure packages, math libraries, etc.)
- Figures are meant to show the significant details at a particular level, not all the details
- Definition of "a significant detail" is solely at the discretion of the designer

Based on these ideas, Figure 20, Figure 21, and Figure 22 explain the meaning of each of the icons available using this notation.

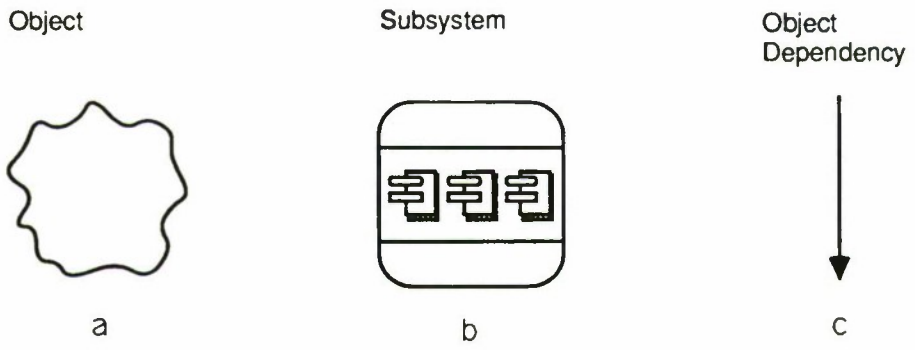


Figure 20: Object, Subsystem, and Dependency Notation

The object (or blob) icon, shown above in Figure 20 (a), represents an identifiable segment of a system, about which there is no implementation information. This icon is not used in this report.

The subsystem icon, shown above in Figure 20 (b), represents a major system component that has a clearly definable interface, yet, which is not representable as a single Ada package. This icon is not used in this report.

The object dependency symbol, shown above in Figure 20 (c), indicates that the object at the origin of the arrow is dependent on the object at the head of the arrow. The origin of the arrow indicates where the dependency occurs. If the origin is in the white area of an icon (shown in subsequent figures), it indicates a specification dependency. If the origin is in the shaded area, it indicates a body dependency.

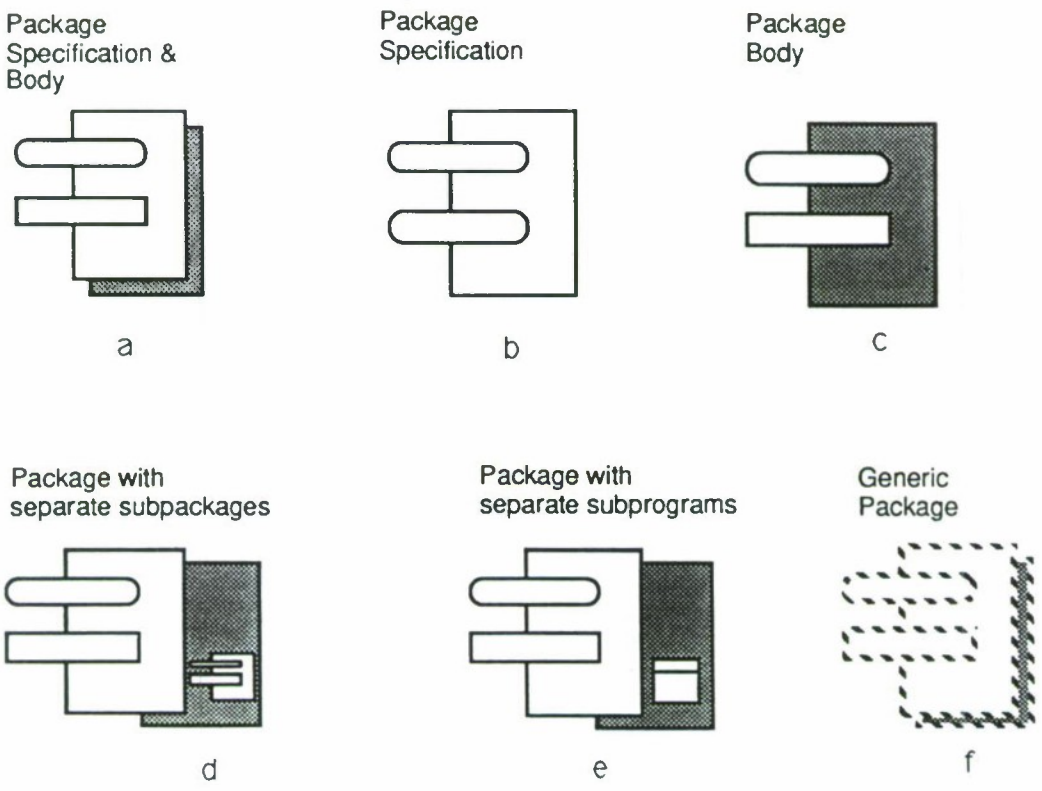


Figure 21: Package Notation

The package specification and body icon, shown above in Figure 21 (a), represents an Ada package specification, the white area, with an associated package body, the shaded area. This icon can be broken apart to show a package specification, Figure 21 (b), or a package body, Figure 21 (c).

Figure 21 (d) and Figure 21 (e) are variations on the package icon which show greater detail. Figure 21 (d) is used to represent packages which have nested subpackages within the body; if the small package icon were placed within the specification, it would indicate visible nested packages. Similarly, Figure 21 (e) illustrates the notation used for separate subprograms within the body of a package.

Finally, Figure 21 (f) illustrates the icon used for generic packages. Everything discussed above in regard to regular packages can also be applied to generic packages.

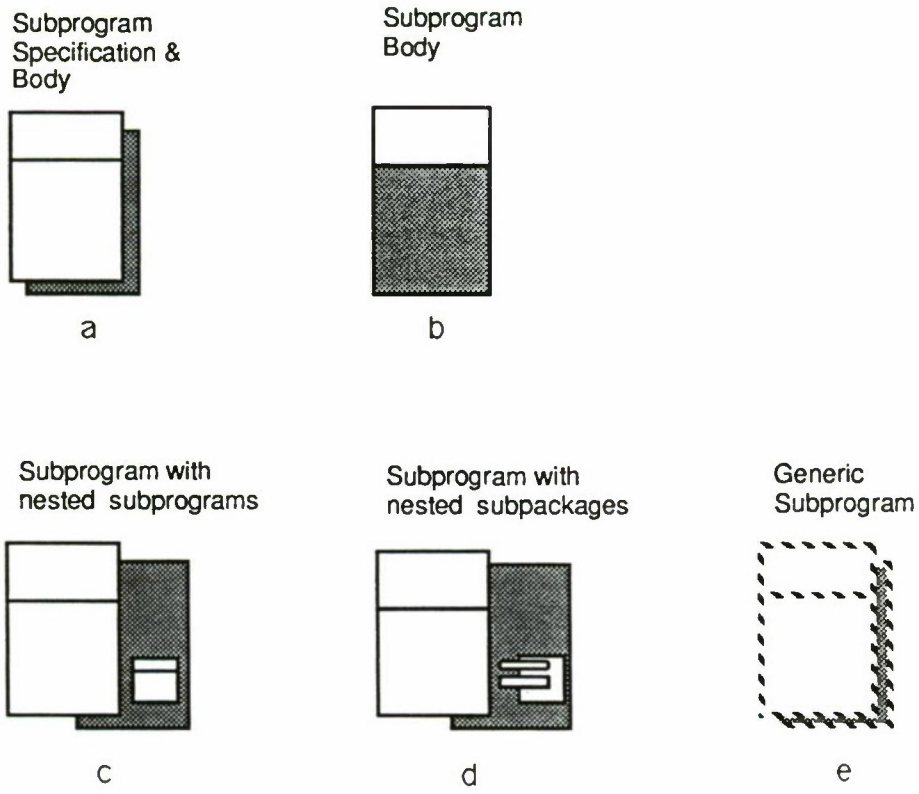


Figure 22: Subprogram Notation

Much of what was discussed previously in regard to packages also applies to subprograms. The subprogram specification and body icon, shown above in Figure 22 (a), represents an Ada subprogram specification and body. The white area represents the specification; the shaded area, the body. This icon can be broken apart to show a separate subprogram body, Figure 22 (b).

Figure 22 (c) and Figure 22 (d) are variations on the subprogram icon and show greater detail. Figure 22 (c) is used to represent subprograms which have nested subprograms within the body. Similarly, Figure 22 (d) illustrates the notation used for separate subpackages within the body of a subprogram.

Finally, Figure 22 (e) illustrates the icon used for generic subprograms. Everything discussed above in regard to regular packages can also be applied to generic subprograms.

Appendix C. Object Manager Template

```
— The following are instructions regarding the use of this template.
— The template does not encompass every procedure,
— however with alterations to the existing
— procedures one can easily affect the neccessary changes.
—
— Perform global substitutes on the following placeholders:
—  {object} gets the name of the object being created
—        ie. {object} => Cb
—
—  {attribute_n} are the attributes of an object you wish to modify.
—        Note : n can take on values 1 to 3
—             ie. {attribute_1} => spark
—
— do a search now for all instances of ?? and fill in the
—   neccessary information

— Finally the user should remove all unwanted code and comments

—|*****************************************************************
—| Module Name:
—|  {object}_Object_Manager
—|
—| Module Type:
—|  Package Specification
—|
—| Module Purpose:
—|  This package implements the type manager for objects
—|  which simulate the electrical system {object}.
—|  This management entails creation of {object} object's
—|  update, maintenance of its state, and state
—|  reporting capabilities.
—|-----------------------------------------------------------------
—| Module Description:
—|  The {object} object manager provides a means to create
—|  an {object} object via the New_{object} operation and returns
—|  an identification for the {object}, which is to be used when
```

```
—| updating/accessing the {object} object's state as described below.
—|
—| The external states of {object} are {attribute_1}, {attribute_2},
—| {attribute_3}, and {attribute_4}.
—| Operations are available to get and set these states.
—|
—| The {object} object manager provides a means to update the
—| state of the object via the:
—|       1) Give_{attribute_1}_To
—|       2) Give_{attribute_2}_To
—|       3) Give_{attribute_3}_To
—|
—| operations, requiring the following external state information:
—|       1) {attribute_1}        {object}_{attribute_1}
—|       2) {attribute_2}        {object}_{attribute_2}
—|       3) {attribute_3}        {object}_{attribute_3}
—|
—| The {object} object manager provides a means of obtaining
—| state information via the:
—|       1) Get_{attribute_1}_From
—|       2) Get_{attribute_2}_From
—|       3) Get_{attribute_3}_From
—|
—| operations, yielding the following internal state information:
—|       1) {attribute_1}        {object}_{attribute_1}
—|       2) {attribute_2}        {object}_{attribute_2}
—|       3) {attribute_3}        {object}_{attribute_3}
—|
—| There are also four functions that are common to every object:
—|
—| procedure Give_Voltage_Lcf_To
—|           (A_{object} : in {object};
—|            A_{object}_Side : in {object}_Side_Names;
—|            Volts : in EU.Voltage;
—|            Load_Conversion : in EU.Load_Conversion_Factor);
—|
—| procedure Give_Current_To (A_{object} : in {object};
—|                            A_{object}_Side : in {object}_Side_Names;
—|                            Load : in EU.Current);
—|
—| procedure Give_Power_Info_To
—|       (A_{object} : in {object};
—|       A_{object}_Side : in {object}_Side_Names;
—|       External_Power_Info : in EU.Power_Info);
—|
—| function Get_Power_Info_From
—|      (A_{object} : in {object};
—|       A_{object}_Side : in {object}_Side_Names)
—|       return EU.Power_Info;
—|
—| Notes:
—|  {object} is an element of an electrical circuit.  Elements are
```

```
--|  connected to connections.  Connections read state information from
--|  an element on one side and write to an element on their other side
--|
--|  One attribute every electrical system object has is Side_Names.
--|  In order to simulate energy flow in a system, the sides
--|  of an object hold the flow through the system to that point:
--|
--|            volts    ---------- volts
--|            ------> X   an  Y ------>
--|            <------ X object Y <------
--|               load ----------   load
--|
--|  The voltage flow through the system to the object is stored at side X.
--|  The load flow through the system to the object is stored at side Y.
--|  The object operation Get_Power_Info_From (A_Side => Y) returns the
--|  value of power_info from side X.
--|
--|*************************************************************

with Electrical_Units;

package {Object}_Object_Manager is

    package Eu renames Electrical_Units;

    type {Object} is private;
    type {Object}_{Attribute_1} is ??;
    type {Object}_{Attribute_2} is ??;
    type {Object}_{Attribute_3} is ??;

    type {Object}_Side_Names is ??:

    function New_{Object}
        ({Attribute_1} : in {Object}_{Attribute_1};
         {Attribute_2} : in {Object}_{Attribute_2};
         {Attribute_3} : in {Object}_{Attribute_3})
         return {Object};
    --|*********************************************************
    --| Description:
    --|  Creates a new {object} as a private type.
    --|  This function returns a pointer to a new {object} object
    --|  representation.  This pointer will be used to access
    --|  the object for state update and state reporting purposes.
    --|
    --| Parameter Description:
    --|  {attribute_1} the default state of the {attribute_1}
    --|  {attribute_2} the default state of the {attribute_2}
    --|  {attribute_3} the default state of the {attribute_3}
    --|  {object} is the access to the private data representaion
    --|
    --|*********************************************************
```

```
procedure Give_Voltage_Lcf_To
          (A_{Object} : In {Object};
           A_{Object}_Side : In {Object}_Side_Names;
           Volts : In Eu.Voltage;
           Load_Conversion : In Eu.Load_Conversion_Factor);
—|*****************************************************************
—| Description:
—|  Places Voltage and LCF on a specific side of a {object}.
—|
—| Parameter Description:
—|  A_{object} is the {object} being acted on.
—|  A_{object}_side is the side of the {object} to be updated
—|  Volts is the Voltage to be given to the {object}
—|  Load_Conversion is the LCF to be given to the {object}
—|*****************************************************************

procedure Give_Current_To (A_{Object} : In {Object};
                           A_{Object}_Side : In {Object}_Side_Names;
                           Load : In Eu.Current);
—|*****************************************************************
—| Description:
—|  Places Current on a specific side of a {object}.
—|
—| Parameter Description:
—|  A_{object} is the {object} being acted on.
—|  A_{object}_side is the side of the {object} to be updated
—|  Current is declared in Electrical_Units
—|*****************************************************************

procedure Give_Power_Info_To
     (A_{Object} : In {Object};
     A_{Object}_Side : In {Object}_Side_Names;
     External_Power_Info : In Eu.Power_Info);
—|*****************************************************************
—| Description:
—|  Places Power_Info on a specific side of a {object}.
—|
—| Parameter Description:
—|  A_{object} is the {object} being acted on.
—|  A_{object}_side is the side of the {object} to be updated
—|  Power_Info is declared in Electrical_Units
—|*****************************************************************

function Get_Power_Info_From
     (A_{Object} : In {Object};
     A_{Object}_Side : In {Object}_Side_Names)
     return Eu.Power_Info;
—|*****************************************************************
—| Description:
—|  Returns Power_Info associated with a specific side of an {object}.
—|
—| Parameter Description:
```

```
—|  A_{object} is the {object} being acted on.
—|  A_{object}_side is the side queried
—|  Power_Info is declared in Electrical_Units
—|**********************************************************************

procedure Give_{Attribute_1}_To
     (A_{Object} : in {Object};
      {Attribute_1} : in {Object}_{Attribute_1});
—|**********************************************************************
—| Description:
—|  Updates the state of {attribute_1} to correspond to current
—|  external conditions
—|
—| Parameter Description:
—|  A_{object} is the {object} to be updated
—|  {attribute_1} is a new {object}_{attribute_1}
—|
—|**********************************************************************

function Get_{Attribute_1}_From
     (A_{Object} : in {Object})
      return {Object}_{Attribute_1};
—|**********************************************************************
—| Description:
—|  Returns the state of {attribute_1}
—|
—| Parameter Description:
—|  A_{object} is the {object} queried
—|  {object}_{attribute_1} is the state of the {attribute_1}
—|
—|**********************************************************************

procedure Give_{Attribute_2}_To
     (A_{Object} : in {Object};
      {Attribute_2} : in {Object}_{Attribute_2});
—|**********************************************************************
—| Description:
—|  Updates the state of {attribute_2} to correspond to current
—|  external conditions
—|
—| Parameter Description:
—|  A_{object} is the {object} to be updated
—|  {attribute_2} is a new {object}_{attribute_2}
—|
—|**********************************************************************

function Get_{Attribute_2}_From
     (A_{Object} : in {Object})
      return {Object}_{Attribute_2};
—|**********************************************************************
—| Description:
—|  Returns the state of {attribute_2}
```

```
    --|
    --| Parameter Description:
    --|   A_{object} is the {object} queried
    --|   {object}_{attribute_2} is the state of the {attribute_2}
    --|
    --|*****************************************************************

    procedure Give_{Attribute_3}_To
        (A_{Object} : in {Object};
         {Attribute_3} : in {Object}_{Attribute_3});
    --|*****************************************************************
    --| Description:
    --|   Updates the state of {attribute_3} to correspond to current
    --|   external conditions
    --|
    --| Parameter Description:
    --|   A_{object} is the {object} to be updated
    --|   {attribute_3} is a new {object}_{attribute_3}
    --|
    --|*****************************************************************

    function Get_{Attribute_3}_From
        (A_{Object} : in {Object})
         return {Object}_{Attribute_3};
    --|*****************************************************************
    --| Description:
    --|   Returns the state of {attribute_3}
    --|
    --| Parameter Description:
    --|   A_{object} is the {object} queried
    --|   {object}_{attribute_3} is the state of the {attribute_3}
    --|
    --|*****************************************************************

pragma Inline (
    New_{Object},
    Give_Voltage_Lcf_To,
    Give_Current_To,
    Give_Power_Info_To,
    Get_Power_Info_From,
    Give_{Attribute_1}_To,
    Get_{Attribute_1}_From
    Give_{Attribute_2}_To,
    Get_{Attribute_2}_From
    Give_{Attribute_3}_To,
    Get_{Attribute_3}_From
    );

private
    type {Object}_Representation;
    -- incomplete type, defined in package body
```

```
    type {Object} is access {Object}_Representation;
    -- pointer to an {object} representation

--|*****************************************************************
--| Modification History:
--|  21Apr87     kl   changed connection arguments to {object} arguments
--|  28Apr87     kl   Updated to include power_info routines
--|  20Mar87     kl   Created
--|
--|-----------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--| The Software engineering Institute (SEI) is a federally funded research
--| and development center established and operated by Carnegie Mellon
--| University (CMU).  Sponsored by the U.S. Department of Defense under
--| contract F19628-85-C-0003, the SEI is supported by the services and
--| defense agencies, with the U.S.  Air Force as the executive contracting
--| agent.
--|
--| Permission to use, copy, modify, or distribute this software and its
--| documentation for any purpose and without fee is hereby granted,
--| provided that the above copyright notice appear in all copies and that
--| both that copyright notice and this permission notice appear in
--| supporting documentation.  Further, the names Software engineering
--| Institute or Carnegie Mellon University may not be used in advertising
--| or publicity pertaining to distribution of the software without
--| specific, written prior permission.  CMU makes no claims or
--| representations about the suitability of this software for any purpose.
--| This software is provided "as is" and no warranty, express or implied,
--| is made by the SEI or CMU, as to the accuracy and functioning of the
--| program and related program material, nor shall the fact of distribution
--| constitute any such warranty.  No responsibility is assumed by the SEI
--| or CMU in connection herewith.
--|*****************************************************************
end {Object}_Object_Manager;
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
pragma Page;
```

```
--|*****************************************************************
--| Module Name:
--|   {object}_Object_Manager
--|
--| Module Type:
--|   Package Body
--|
--|-----------------------------------------------------------------
--| Module Description:
--|   Manipulates private data structures that represent
--|   a {object}, and contact points on a {object}.
--|
--| Notes:
--|        none
--|*****************************************************************

package body {Object}_Object_Manager is

    type Point_Representation is array ({Object}_Side_Names) of Eu.Power_Info;

    -- representation of a {object}
    --
    type {Object}_Representation is
        record
            Points: Point_Representation;
            {Attribute_1}: {Object}_{Attribute_1}:= ??;
            {Attribute_2}: {Object}_{Attribute_2}:= ??;
            {Attribute_2}: {Object}_{Attribute_3}:= ??;
        end record;

    pragma page;
```

```
function Opposite_Side (This_Side: in {Object}_Side_Names)
                        return {Object}_Side_Names is
   —|*************************************************************
   —| Description:
   —|  A function to find the opposite side of a {object}
   —|
   —| Parameter Description:
   —|  This_Side is the side for which the opposite is sought.
   —|  Side_Names is the opposite side.
   —|
   —| Notes:
   —|  USED FOR CONTROL ELEMENTS SUCH AS {object}S, RELAYS AND
   —|  SWITCHES.
   —|*************************************************************

   The_Side: {Object}_Side_Names:= ??; — one of the sides

begin
   — select opposite side based on what this side is.
   —
   If This_Side = ?? then
      The_Side:= ??; — the other side
   end If;

   RETURN The_Side;

end Opposite_Side;

pragma Page;
```

```
function New_{Object}
      ({Attribute_1} : in {Object}_{Attribute_1};
       {Attribute_2} : in {Object}_{Attribute_2};
       {Attribute_3} : in {Object}_{Attribute_3})
       return {Object} is
—|*****************************************************************
—| Description:
—|  Creates a new {object} as a private type.
—|  This function returns a pointer to a new {object} object
—|  representation.  This pointer will be used to identify
—|  the object for state update and state reporting purposes.
—|
—| Parameter Description:
—|  {attribute_1} the default state of the {attribute_1}
—|  {attribute_2} the default state of the {attribute_2}
—|  {attribute_3} the default state of the {attribute_3}
—|  {object} is the access to the private data representaion
—|
—|*****************************************************************

    The_New_Object: {Object}:= new {Object}_Representation;

begin
    The_New_Object.{Attribute_1}:= {Attribute_1};
    The_New_Object.{Attribute_2}:= {Attribute_2};
    The_New_Object.{Attribute_3}:= {Attribute_3};

    RETURN The_New_Object;
end New_{Object};

pragma page;
```

```
procedure Give_Voltage_Lcf_To
          (A_{Object} : in {Object};
           A_{Object}_Side : in {Object}_Side_Names;
           Volts : in Eu.Voltage;
           Load_Conversion : in Eu.Load_Conversion_Factor) is
—|*****************************************************************
—| Description:
—|   Places Voltage and LCF on a specific side of a {object}.
—|
—| Parameter Description:
—|   A_{object} is the {object} being acted on.
—|   A_{object}_side is the side of the {object} to be updated
—|   Volts is the Voltage to be given to the {object}
—|   Load_Conversion is the LCF to be given to the {object}
—|*****************************************************************
begin
     A_{Object}.Points (A_{Object}_Side).V:= Volts;
     A_{Object}.Points (A_{Object}_Side).Lcf:= Load_Conversion;
end Give_Voltage_Lcf_To;

pragma Page;
```

```
procedure Give_Current_To (A_{Object} : in {Object};
                           A_{Object}_Side : in {Object}_Side_Names;
                           Load : in Eu.Current) is
--/*****************************************************************
--| Description:
--|   Places Current on a specific side of a {object}.
--|
--| Parameter Description:
--|   A_{object} is the {object} being acted on.
--|   A_{object}_side is the side of the {object} to be updated
--|   Current is declared in Electrical_Units
--/*****************************************************************
begin
    A_{Object}.Points (A_{Object}_Side).I:= Load;
end Give_Current_To;

pragma Page;
```

```
procedure Give_Power_Info_To
      (A_{Object} : in {Object};
      A_{Object}_Side : in {Object}_Side_Names;
      External_Power_Info : in Eu.Power_Info) is
--|*****************************************************************
--| Description:
--|   Places Power_Info on a specific side of a {object}.
--|
--| Parameter Description:
--|   A_{object} is the {object} being acted on.
--|   A_{object}_side is the side of the {object} to be updated
--|   Power_Info is declared in Electrical_Units
--|*****************************************************************
begin
     A_{Object}.Points (A_{Object}_Side):= External_Power_Info;

end Give_Power_Info_To;

pragma Page;
```

```
function Get_Power_Info_From
     (A_{Object} : In {Object};
      A_{Object}_Side : In {Object}_Side_Names)
      return Eu.Power_Info Is
--|*****************************************************************
--| Description:
--|   Returns Power_Info associated with
--|   a specific side of an {object}.
--|
--| Parameter Description:
--|   A_{object} is the {object} being acted on.
--|   A_{object}_side is the side queried
--|   Power_Info is declared in Electrical_Units
--|*****************************************************************
     The_Power_Info: Eu.Power_Info;

begin
     If A_{Object}.{Attribute_1} = Closed then
          The_Power_Info:= A_{Object}.Points (Opposite_Side (A_{Object}_Side));
     end If;

     RETURN The_Power_Info;
end Get_Power_Info_From;

pragma Page;
```

```
procedure Give_{Attribute_1}_To
     (A_{Object} : in {Object};
      {Attribute_1} : in {Object}_{Attribute_1}) is
—|*****************************************************************
—| Description:
—|   Updates the state of {attribute_1} to correspond to current
—|   external conditions
—|
—| Parameter Description:
—|   A_{object} is the {object} to be updated
—|   {attribute_1} is a new {object}_{attribute_1}
—|
—|*****************************************************************
begin
     A_{Object}.{Attribute_1}:= {Attribute_1};
end Give_{Attribute_1}_To;

pragma Page;
```

```
function Get_{Attribute_1}_From
     (A_{Object} : in {Object})
      return {Object}_{Attribute_1} is
—|*****************************************************************
—| Description:
—|   Returns the state of {attribute_1}
—|
—| Parameter Description:
—|   A_{object} is the {object} queried
—|   {object}_{attribute_1} is the state of the {attribute_1}
—|
—|*****************************************************************
begin
     RETURN A_{Object}.{Attribute_1};
end Get_{Attribute_1}_From;

pragma Page;
```

```
procedure Give_{Attribute_2}_To
     (A_{Object} : in {Object};
      {Attribute_2} : in {Object}_{Attribute_2}) is
—|*****************************************************************
—| Description:
—|   Updates the state of {attribute_2} to correspond to current
—|   external conditions
—|
—| Parameter Description:
—|   A_{object} is the {object} to be updated
—|   {attribute_2} is a new {object}_{attribute_2}
—|
—|*****************************************************************
begin
     A_{Object}.{Attribute_2}:= {Attribute_2};
end Give_{Attribute_2}_To;

pragma Page;
```

```
function Get_{Attribute_2}_From
     (A_{Object} : in {Object})
      return {Object}_{Attribute_2} is
--|*****************************************************************
--| Description:
--|   Returns the state of {attribute_2}
--|
--| Parameter Description:
--|   A_{object} is the {object} queried
--|   {object}_{attribute_2} is the state of the {attribute_2}
--|
--|*****************************************************************
begin
     RETURN A_{Object}.{Attribute_2};
end Get_{Attribute_2}_From;

pragma Page;
```

```
procedure Give_{Attribute_3}_To
    (A_{Object} : in {Object};
     {Attribute_3} : in {Object}_{Attribute_3}) is
--|*****************************************************************
--| Description:
--|   Updates the state of {attribute_3} to correspond to current
--|   external conditions
--|
--| Parameter Description:
--|   A_{object} is the {object} to be updated
--|   {attribute_3} is a new {object}_{attribute_3}
--|
--|*****************************************************************
begin
    A_{Object}.{Attribute_3}:= {Attribute_3};
end Give_{Attribute_3}_To;

pragma Page;
```

```
    function Get_{Attribute_3}_From
        (A_{Object} : in {Object})
         return {Object}_{Attribute_3} is
    —|*****************************************************************
    —| Description:
    —|   Returns the state of {attribute_3}
    —|
    —| Parameter Description:
    —|   A_{object} is the {object} queried
    —|   {object}_{attribute_3} is the state of the {attribute_3}
    —|
    —|*****************************************************************
    begin
        RETURN A_{Object}.{Attribute_3};
    end Get_{Attribute_3}_From;

—|*********************************************************************
—| Modification History:
—| 30Apr87  kl   Updated read routines to reflect operation of an {object}.
—| 28Apr87  kl   Added power_info routines
—| 20Mar87  kl   Created
—|
—|---------------------------------------------------------------------
—| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
—| The Software engineering Institute (SEI) is a federally funded research
—| and development center established and operated by Carnegie Mellon
—| University (CMU). Sponsored by the U.S. Department of Defense under
—| contract F19628–85–C–0003, the SEI is supported by the services and
—| defense agencies, with the U.S. Air Force as the executive contracting
—| agent.
—|
—| Permission to use, copy, modify, or distribute this software and its
—| documentation for any purpose and without fee is hereby granted,
—| provided that the above copyright notice appear in all copies and that
—| both that copyright notice and this permission notice appear in
—| supporting documentation. Further, the names Software engineering
—| Institute or Carnegie Mellon University may not be used in advertising
—| or publicity pertaining to distribution of the software without
—| specific, written prior permission. CMU makes no claims or
—| representations about the suitability of this software for any purpose.
—| This software is provided "as is" and no warranty, express or implied,
—| is made by the SEI or CMU, as to the accuracy and functioning of the
—| program and related program material, nor shall the fact of distribution
—| constitute any such warranty. No responsibility is assumed by the SEI
—| or CMU in connection herewith.
—|*********************************************************************
end {Object}_Object_Manager;
```

Appendix D. DC Power System Ada Code

The Ada code that follows implements a subset of a typical aircraft simulator DC power system. The implementation includes package specifications, bodies and a test driver. The following disclaimer applies to all the code in this appendix and throughout the rest of this report:

D.1 Package Electrical_Units

```
--|*****************************************************************
--| Module Name:
--|   Electrical_Units
--|
--| Module Type:
--|   Package Specification
--|
--| Module Purpose:
--|   This package provides basic electrical types: current, voltage and load
--|   conversion factor (LCF). It also declares common constant values
--|   for those types. Also provides the agregate record power_info,
--|   which contains fields for voltage, current, and LCF.
--|-----------------------------------------------------------------
--| Module Description:
--|   Voltage is an enumerated type.  Current and LCFs
--|   are real values.
--|
--| Notes:
--|   This package has no body.
--|*****************************************************************

package Electrical_Units is

    type Voltage is (Floating_Voltage,
                     Zero_Voltage,
                     Available_Voltage);
    No_Voltage : constant Voltage := Zero_Voltage;

    -- Devices like relays need to know if
    -- voltage is available without concern for the level.
    --
    Energizing_Voltage : constant Voltage := Available_Voltage;

    type Current is new Float;
    No_Current : constant Current := 0.0;

    type Load_Conversion_Factor is new Float;
    No_Load_Conversion : constant Load_Conversion_Factor := 0.0;

    --  Needed when device shorts out when current passes the wrong way.
    --
    Max_Load_Conversion : constant Load_Conversion_Factor := 10_000.0;

    --  Permits a function to return all three values.
    --
    type Power_Info is
        record
            V : Voltage := Floating_Voltage;
            I : Current := No_Current;
```

```
            Lcf : Load_Conversion_Factor := No_Load_Conversion;
        end record;

--|*****************************************************************
--| Modification History:
--|  05Oct87     tac  Deleted body to this spec. Deleted function "*".
--|  02May87     kl added function "*"
--|  23Apr87     kl added floating voltage; made it default for power_info
--|  03Mar87     mmr added power_info
--|  27Feb87     kl added constants
--|  17Feb87     kl initial version
--|-----------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*****************************************************************
end Electrical_Units;

pragma Page;
```

D.2 Package Global_Types

```
--|*****************************************************************
--| Module Name:
--|   Global Types
--|
--| Module Type:
--|   Package Specification
--|
--| Module Purpose:
--|   This package provides global types for use throughout the simulator code.
--|-----------------------------------------------------------------
--| Module Description:
--|   This package provides global types for use throughout the
--|   simulator code.
--|
--|   Type Execution_Sequence defines the frames to be used by the
--|   executives during the cyclic execution of the code.
--|
--| Notes:
--|   This package has no body.
--|*****************************************************************

package Global_Types is

    type Execution_Sequence is (Frame_1_Modules_Are_Executed,
                                Frame_2_Modules_Are_Executed,
                                Frame_3_Modules_Are_Executed,
                                Frame_4_Modules_Are_Executed,
                                Frame_5_Modules_Are_Executed,
                                Frame_6_Modules_Are_Executed,
                                Frame_7_Modules_Are_Executed,
                                Frame_8_Modules_Are_Executed);

    type Element_Class is (A_Cb, A_Tru, A_Bus);
--|*****************************************************************
--| Modification History:
--|   28Feb89   kl   changed "wire" to "bus"
--|   14Nov88   kl   removed wire_point, object_point
--|   17Jun88   der      Added constants wire_point, object_point.
--|   14Jun88   der      A_Wire added to Element_Class.
--|   09Oct87   tac  Moved Element_Class into this package from the aggregates.
--|   24Apr87   kl   created
--|
--|-----------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*****************************************************************
end Global_Types;
```

D.3 Package Flight_System_Names

```
--|*****************************************************************
--| Module Name:
--|   Flight System Names
--|
--| Module Type:
--|   Package Specification
--|
--| Module Purpose:
--|   This package names all systems under the flight executive
--|-----------------------------------------------------------------
--| Module Description:
--|   This package names all systems under the flight executive
--|
--| Notes:
--|   This package has no body.
--|*****************************************************************

package Flight_System_Names is

    type Name_Of_A_Flight_System is (
        Dc_Power,
        Ac_Power,
        Engine_1,
        Engine_2,
        Engine_3,
        Engine_4,
        Dummy);

--|*****************************************************************
--| Modification History:
--|   30Sep87     tac   added dummy to System names
--|   21Aug87     kl    created
--|-----------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*****************************************************************
end Flight_System_Names;

pragma Page;
```

D.4 Package Cb_Object_Manager

```
—|**********************************************************
—| Module Name:
—|   Cb_Object_Manager
—|
—| Module Type:
—|   Package Specification
—|
—| Module Purpose:
—|   This package implements the type manager for objects
—|   which simulate the Electrical system circuit breaker (Cb).
—|   This management entails creation of Cb objects,
—|   update, maintenance of its state, and state
—|   reporting capabilities.
—|----------------------------------------------------------
—| Module Description:
—|   The  Cb_Object_Manager provides a means to create
—|   a Cb object via the New_Cb operation and returns
—|   an identification for the Cb, which is to be used when
—|   updating/accessing the Cb object's state as described below.
—|
—|   The external states of Cb are Position and Rating
—|   Operations are available to get and set these states.
—|
—|   The Cb_Object_Manager provides a means to update the
—|   state of the object via the:
—|         1) Give_Position_To
—|   operation, requiring the following external state information:
—|         1) Position: Cb_Position
—|
—|   The Cb_Object_Manager provides a means of obtaining
—|   state information via the:
—|         1) Get_Position_From
—|   operation, yielding the following internal state information:
—|         1) Position: Cb_Position
—|
—|   There are also four functions that are common to every object:
—|
—|   procedure Give_Voltage_Lcf_To (A_Cb : in Cb;
—|                 A_Cb_Side : in Cb_Side_Names;
—|                 Volts : in EU.Voltage;
—|                 Load_Conversion : in EU.Load_Conversion_Factor);
—|
—|   procedure Give_Current_To (A_Cb : in Cb;
—|                              A_Cb_Side : in Cb_Side_Names;
—|                              Load : in EU.Current);
—|
—|   procedure Give_Power_Info_To (A_Cb : in Cb;
```

```
—|        A_Cb_Side : in Cb_Side_Names;
—|        External_Power_Info : in EU.Power_Info);
—|
—| function Get_Power_Info_From (A_Cb : in Cb;
—|        A_Cb_Side : in Cb_Side_Names)
—|        return EU.Power_Info;
—|
—| Notes:
—|  Cb is an element of an electrical circuit.  Elements are
—|  connected to connections.  Connections read state information from
—|  an element on one side and write to an element on their other side
—|
—|  One attribute every electrical system object has is Side_Names.
—|  In order to simulate energy flow in a system, the sides
—|  of an object hold the flow through the system to that point:
—|
—|              volts   ————————  volts
—|              ———>  X  an  Y  ———>
—|
—|                load ————————   load
—|
—|  The voltage flow through the system to the object is stored at side X
—|  The load flow through the system to the object is stored at side Y.
—|  The object operation Get_Power_Info_From (a_side => Y) returns the
—| value of power_info from side X.
—|
—|*****************************************************************

with Electrical_Units;

package Cb_Object_Manager is

    package Eu renames Electrical_Units;

    type Cb is private;
    type Cb_Position is (Open, Closed);
    type Cb_Rating is new Float;

    type Cb_Side_Names is (Side_1, Side_2);

    function New_Cb (Position : in Cb_Position;
          Rating : in Cb_Rating)
          return Cb;
    —|*****************************************************************
    —| Description:
    —|  Creates a new Cb as a private type.
    —|  This function returns a pointer to a new Cb_Representation.
    —| This pointer will be used to access
    —|  the object for state update and state reporting purposes.
    —|
    —| Parameter Description:
    —|  Position the default state of the Position
```

```
--| Rating the default state of the Rating
--| Cb is the access to the private data representaion
--|
--|*****************************************************************

procedure Give_Voltage_Lcf_To (A_Cb : in Cb;
          A_Cb_Side : in Cb_Side_Names;
          Volts : in Eu.Voltage;
          Load_Conversion : in Eu.Load_Conversion_Factor);
--|*****************************************************************
--| Description:
--| Places Voltage and LCF on a specific side of a Cb.
--|
--| Parameter Description:
--| A_Cb is the Cb being acted on.
--| A_Cb_Side is the side of the Cb to be updated
--| Volts is the Voltage to be given to the Cb
--| Load_Conversion is the LCF to be given to the Cb
--|*****************************************************************

procedure Give_Current_To (A_Cb : in Cb;
                           A_Cb_Side : in Cb_Side_Names;
                           Load : in Eu.Current);
--|*****************************************************************
--| Description:
--| Places Current on a specific side of a Cb.
--|
--| Parameter Description:
--| A_Cb is the Cb being acted on.
--| A_Cb_Side is the side of the Cb to be updated
--| Current is declared in Electrical_Units
--|*****************************************************************

procedure Give_Power_Info_To (A_Cb : in Cb;
      A_Cb_Side : in Cb_Side_Names;
      External_Power_Info : in Eu.Power_Info);
--|*****************************************************************
--| Description:
--| Places Power_Info on a specific side of a Cb.
--|
--| Parameter Description:
--| A_Cb is the Cb being acted on.
--| A_Cb_Side is the side of the Cb to be updated
--| Power_Info is declared in Electrical_Units
--|*****************************************************************

function Get_Power_Info_From (A_Cb : in Cb;
     A_Cb_Side : in Cb_Side_Names)
     return Eu.Power_Info;
--|*****************************************************************
--| Description:
--| Returns Power_Info associated with a specific side of a Cb
```

```
    --|
    --| Parameter Description:
    --|   A_Cb is the Cb being acted on.
    --|   A_Cb_Side is the side queried
    --|   Power_Info is declared in Electrical_Units
    --|*****************************************************************

    procedure Give_Position_To (A_Cb : In Cb;
        Position : In Cb_Position);
    --|*****************************************************************
    --| Description:
    --|   Updates the state of Position to correspond to current
    --|   external conditions
    --|
    --| Parameter Description:
    --|   A_Cb is the Cb to be updated
    --|   Position is a new Cb_Position
    --|
    --|*****************************************************************

    function Get_Position_From (A_Cb : In Cb) return Cb_Position;
    --|*****************************************************************
    --| Description:
    --|   Returns the state of Position
    --|
    --| Parameter Description:
    --|   A_Cb is the Cb queried
    --|   Cb_Position is the state of the Position
    --|
    --|*****************************************************************

pragma Inline (
    New_Cb,
    Give_Voltage_Lcf_To,
    Give_Current_To,
    Give_Power_Info_To,
    Get_Power_Info_From,
    Give_Position_To,
    Get_Position_From
    );

private
    type Cb_Representation;        -- incomplete type, defined in package body
    type Cb is access Cb_Representation; -- pointer to an Cb representation

--|*****************************************************************
--| Modification History:
--|  21Apr87    kl   changed connection arguments to Cb arguments
--|  28Apr87    kl   Updated to include power_info routines
--|  20Mar87    kl   Created
--|
--|-----------------------------------------------------------------
```

```
—| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
—|**********************************************************************
end Cb_Object_Manager;

pragma Page;
```

D.5 Package Body Cb_Object_Manager

```
--|*****************************************************************
--| Module Name:
--|   Cb_Object_Manager
--|
--| Module Type:
--|   Package Body
--|
--|-----------------------------------------------------------------
--| Module Description:
--|   Manipulates private data structures that represent
--|   a circuit breaker (Cb), and contact points on a Cb.
--|
--| Notes:
--|       none
--|*****************************************************************

package body Cb_Object_Manager is

    type Point_Representation is array (Cb_Side_Names) of Eu.Power_Info;

    -- representation of a Cb
    --
    type Cb_Representation is
        record
            Points: Point_Representation;
            Position: Cb_Position:= Closed;
            Rating: Cb_Rating:= 50.0;
        end record;

pragma Page;;
```

```
function Opposite_Side (This_Side: in Cb_Side_Names) return Cb_Side_Names is
    --|*************************************************************
    --| Description:
    --|   A function to find the opposite side of a Cb
    --|
    --| Parameter Description:
    --|   This_Side is the side for which the opposite is sought.
    --|   Side_Names is the opposite side.
    --|
    --| Notes:
    --|   USED FOR CONTROL ELEMENTS SUCH AS CbS, RELAYS AND
    --|   SWITCHES.
    --|*************************************************************

    The_Side: Cb_Side_Names:= Side_1; -- one of the sides

    begin
        -- select opposite side based on what this side is.
        --
        if This_Side = Side_1 then
            The_Side:= Side_2; -- the other side
        end if;

        RETURN The_Side;

end Opposite_Side;

pragma Page;;
```

```
function New_Cb (Position : In Cb_Position;
                 Rating : In Cb_Rating)
                 return Cb Is
    --|*****************************************************************
    --| Description:
    --|   Creates a new Cb as a private type.
    --|   This function returns a pointer to a new Cb object
    --|   representation.  This pointer will be used to access
    --|   the object for state update and state reporting purposes.
    --|
    --| Parameter Description:
    --|   Position the default state of the Position
    --|   Rating the default state of the Rating
    --|   Cb is the access to the private data representaion
    --|*****************************************************************

    The_New_Object: Cb:= new Cb_Representation;

    begin
        The_New_Object.Position:= Position;
        The_New_Object.Rating:= Rating;

        RETURN The_New_Object;
end New_Cb;

pragma Page;
```

```
procedure Give_Voltage_Lcf_To (A_Cb : In Cb;
                A_Cb_Side : In Cb_Side_Names;
                Volts : In Eu.Voltage;
                Load_Conversion : In Eu.Load_Conversion_Factor) Is
    --|**********************************************************
    --| Description:
    --|   Places Voltage and LCF on a specific side of a Cb.
    --|
    --| Parameter Description:
    --|   A_Cb is the Cb being acted on.
    --|   A_Cb_Side is the side of the Cb to be updated
    --|   Volts is the Voltage to be given to the Cb
    --|   Load_Conversion is the LCF to be given to the Cb
    --|**********************************************************
    begin
        A_Cb.Points (A_Cb_Side).V:= Volts;
        A_Cb.Points (A_Cb_Side).Lcf:= Load_Conversion;
end Give_Voltage_Lcf_To;

pragma Page;
```

```
procedure Give_Current_To (A_Cb : In Cb;
                           A_Cb_Side : In Cb_Side_Names;
                           Load : In Eu.Current) Is
    --|*****************************************************************
    --| Description:
    --|   Places Current on a specific side of a Cb.
    --|
    --| Parameter Description:
    --|   A_Cb is the Cb being acted on.
    --|   A_Cb_Side is the side of the Cb to be updated
    --|   Current is declared in Electrical_Units
    --|*****************************************************************
    begin
        A_Cb.Points (A_Cb_Side).I:= Load;
end Give_Current_To;

pragma Page;
```

```
procedure Give_Power_Info_To (A_Cb : In Cb;
          A_Cb_Side : In Cb_Side_Names;
          External_Power_Info : In Eu.Power_Info) Is
    --|*****************************************************************
    --| Description:
    --|   Places Power_Info on a specific side of a Cb.
    --|
    --| Parameter Description:
    --|   A_Cb is the Cb being acted on.
    --|   A_Cb_Side is the side of the Cb to be updated
    --|   Power_Info is declared in Electrical_Units
    --|*****************************************************************
    begin
        A_Cb.Points (A_Cb_Side):= External_Power_Info;
end Give_Power_Info_To;

pragma Page;
```

```
function Get_Power_Info_From (A_Cb : In Cb;
        A_Cb_Side : In Cb_Side_Names)
        return Eu.Power_Info is
    --/*****************************************************************
    --| Description:
    --|  Returns Power_Info associated with a specific side of an Cb.
    --|
    --| Parameter Description:
    --|  A_Cb is the Cb being acted on.
    --|  A_Cb_Side is the side queried
    --|  Power_Info is declared in Electrical_Units
    --/*****************************************************************
    The_Power_Info: Eu.Power_Info;

    begin
        If A_Cb.Position = Closed then
            The_Power_Info:= A_Cb.Points (Opposite_Side (A_Cb_Side));
        end If;

        RETURN The_Power_Info;
end Get_Power_Info_From;

pragma Page;
```

```
procedure Give_Position_To (A_Cb : In Cb;
          Position : In Cb_Position) Is
    --|**********************************************************
    --| Description:
    --|   Updates the state of Position to correspond to current
    --|   external conditions
    --|
    --| Parameter Description:
    --|   A_Cb is the Cb to be updated
    --|   Position is a new Cb_Position
    --|
    --|**********************************************************
    begin
        A_Cb.Position:= Position;
end Give_Position_To;

pragma Page;
```

```
function Get_Position_From (A_Cb : in Cb) return Cb_Position is
    --|*****************************************************************
    --| Description:
    --|   Returns the state of Position
    --|
    --| Parameter Description:
    --|   A_Cb is the Cb queried
    --|   Cb_Position is the state of the Position
    --|
    --|*****************************************************************
    begin
        RETURN A_Cb.Position;
end Get_Position_From;

--|*****************************************************************
--| Modification History:
--|  05Oct87      tac   Replaced all occurances of "connection" with
--|                     "Point".
--|  30Apr87 kl   Updated read routines to reflect operation of a Cb.
--|  28Apr87 kl   Added power_info routines
--|  20Mar87kl    Created
--|
--|-----------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*****************************************************************
end Cb_Object_Manager;

pragma Page;
```

D.6 Package Tru_Object_Manager

```
—|**********************************************************
—| Module Name:
—|   Tru_Object_Manager
—|
—| Module Type:
—|   Package Specification
—|
—| Module Purpose:
—|   This package implements the type manager for objects
—|   which simulate the electrical system Tru.
—|   This management entails creation of Tru objects,
—|   update, maintenance of its state, and state reporting capabilities.
—|----------------------------------------------------------
—| Module Description:
—|   The  Tru_Object_Manager provides a means to create
—|   an Tru object via the New_Tru operation and returns
—|   an identification for the Tru, which is to be used when
—|   updating/accessing the Tru object's state as described below.
—|
—|   There are four functions that are common to every object:
—|
—|  procedure Give_Voltage_Lcf_To (A_Tru : in Tru;
—|              A_Tru_Side : in Tru_Side_Names;
—|              Volts : in EU.Voltage;
—|              Load_Conversion : in EU.Load_Conversion_Factor);
—|
—|  procedure Give_Current_To (A_Tru : in Tru;
—|                             A_Tru_Side : in Tru_Side_Names;
—|                             Load : in EU.Current);
—|
—|  procedure Give_Power_Info_To (A_Tru : in Tru;
—|          A_Tru_Side : in Tru_Side_Names;
—|          External_Power_Info : in EU.Power_Info);
—|
—|  function Get_Power_Info_From (A_Tru : in Tru;
—|         A_Tru_Side : in Tru_Side_Names)
—|         return EU.Power_Info;
—|
—| Notes:
—|   Tru is an element of an electrical circuit.  Elements are
—|   connected to connections.  Connections read state information from
—|   an element on one side and write to an element on their other side
—|
—|   One attribute every electrical system object has is Side_Names.
—|   In order to simulate energy flow in a system, the sides
—|   of an object hold the flow through the system to that point:
—|
```

```
--|            volts  ---------  volts
--|            ------>  X  an  Y  ------>
--|
--|              load ---------   load
--|
--|  The voltage flow through the system to the object is stored at side X
--|  The load flow through the system to the object is stored at side Y.
--|  The object operation Get_Power_Info_From (a_side => Y) returns the
--| value of power_info from side X.
--|
--|*************************************************************

with Electrical_Units;

package Tru_Object_Manager is

    package Eu renames Electrical_Units;

    type Tru is private;
    type Tru_Side_Names is (Ac_Side, Dc_Side);

    function New_Tru (Lcf : in Eu.Load_Conversion_Factor;
          Load : in Eu.Current)
          return Tru;
    --|*********************************************************
    --| Description:
    --|  Creates a new Tru as a private type.
    --|  This function returns a pointer to a new Tru object
    --|  representation.  This pointer will be used to access
    --|  the object for state update and state reporting purposes.
    --|
    --| Parameter Description:
    --|  Lcf the default state of the Lcf
    --|  Load the default state of the Load
    --|  Tru is the access to the private data representaion
    --|
    --|*********************************************************

    procedure Give_Voltage_Lcf_To (A_Tru : in Tru;
              A_Tru_Side : in Tru_Side_Names;
              Volts : in Eu.Voltage;
              Load_Conversion : in Eu.Load_Conversion_Factor);
    --|*********************************************************
    --| Description:
    --|  Places Voltage and LCF on a specific side of a Tru.
    --|
    --| Parameter Description:
    --|  A_Tru is the Tru being acted on.
    --|  A_Tru_Side is the side of the Tru to be updated
    --|  Volts is the Voltage to be given to the Tru
    --|  Load_Conversion is the LCF to be given to the Tru
    --|*********************************************************
```

```
    procedure Give_Current_To (A_Tru : in Tru;
                               A_Tru_Side : in Tru_Side_Names;
                               Load : in Eu.Current);
    --|*************************************************************
    --| Description:
    --|   Places Current on a specific side of a Tru.
    --|
    --| Parameter Description:
    --|   A_Tru is the Tru being acted on.
    --|   A_Tru_Side is the side of the Tru to be updated
    --|   Current is declared in Electrical_Units
    --|*************************************************************

    procedure Give_Power_Info_To (A_Tru : in Tru;
          A_Tru_Side : in Tru_Side_Names;
          External_Power_Info : in Eu.Power_Info);
    --|*************************************************************
    --| Description:
    --|   Places Power_Info on a specific side of a Tru.
    --|
    --| Parameter Description:
    --|   A_Tru is the Tru being acted on.
    --|   A_Tru_Side is the side of the Tru to be updated
    --|   Power_Info is declared in Electrical_Units
    --|*************************************************************

    function Get_Power_Info_From (A_Tru : in Tru;
          A_Tru_Side : in Tru_Side_Names)
          return Eu.Power_Info;
    --|*************************************************************
    --| Description:
    --|   Returns Power_Info associated with a specific side of an Tru.
    --|
    --| Parameter Description:
    --|   A_Tru is the Tru being acted on.
    --|   A_Tru_Side is the side queried
    --|   Power_Info is declared in Electrical_Units
    --|*************************************************************

pragma Inline (
    New_Tru,
    Give_Voltage_Lcf_To,
    Give_Current_To,
    Give_Power_Info_To,
    Get_Power_Info_From
    );

private
    type Tru_Representation; -- incomplete type, defined in package body
    type Tru is access Tru_Representation;     -- pointer to an Tru representation
```

```
--|*****************************************************************
--| Modification History:
--|  21Apr87    kl   changed connection arguments to Tru arguments
--|  28Apr87    kl   Updated to include power_info routines
--|  20Mar87    kl   Created
--|
--|-----------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*****************************************************************
end Tru_Object_Manager;

pragma Page;
```

D.7 Package Body Tru_Object_Manager

```
--/*****************************************************************
--| Module Name:
--|   Tru_Object_Manager
--|
--| Module Type:
--|   Package Body
--|
--|---------------------------------------------------------------
--| Module Description:
--|   Manipulates private data structures that represent
--|   a Tru, and contact points on a Tru.
--|
--| Notes:
--|        none
--|*****************************************************************

package body Tru_Object_Manager is

    type Point_Representation is array (Tru_Side_Names) of Eu.Power_Info;

    -- representation of a Tru
    --
    type Tru_Representation is
        record
            Points: Point_Representation;
            Lcf: Eu.Load_Conversion_Factor:= Eu.No_Load_Conversion;
            Load: Eu.Current:= Eu.No_Current;
        end record;

    function "+"(A_Current: in Eu.Current;
                 Another_Current: in Eu.Current)
                 return Eu.Current renames Eu."+";

pragma Page;
```

```
function Opposite_Side (This_Side: In Tru_Side_Names) return Tru_Side_Names is
    --|*************************************************************
    --| Description:
    --|   A function to find the opposite side of a Tru
    --|
    --| Parameter Description:
    --|   This_Side is the side for which the opposite is sought.
    --|   Side_Names is the opposite side.
    --|
    --| Notes:
    --|   USED FOR CONTROL ELEMENTS SUCH AS TruS, RELAYS AND
    --|   SWITCHES.
    --|*************************************************************
    The_Side : Tru_Side_Names := Ac_Side; -- one of the sides

    begin
        --
        -- select opposite side based on what this side is.
        --
        If This_Side = Ac_Side then
            The_Side := Dc_Side;
        end If;

        RETURN The_Side;
end Opposite_Side;

pragma Page;
```

```
function New_Tru (Lcf : In Eu.Load_Conversion_Factor;
                  Load : In Eu.Current)
                  return Tru is
    --|*************************************************************
    --| Description:
    --|   Creates a new Tru as a private type.
    --|   This function returns a pointer to a new Tru object
    --|   representation.  This pointer will be used to access
    --|   the object for state update and state reporting purposes.
    --|
    --| Parameter Description:
    --|   Lcf the default state of the Lcf
    --|   Load the default state of the Load
    --|   Tru is the access to the private data representaion
    --|*************************************************************

    The_New_Object: Tru:= new Tru_Representation;

    begin
        The_New_Object.Lcf:= Lcf;
        The_New_Object.Load:= Load;

        RETURN The_New_Object;
end New_Tru;

pragma Page;
```

```
procedure Give_Voltage_Lcf_To (A_Tru : In Tru;
                A_Tru_Side : In Tru_Side_Names;
                Volts : In Eu.Voltage;
                Load_Conversion : In Eu.Load_Conversion_Factor) Is
    --|*****************************************************************
    --| Description:
    --|   Places Voltage and LCF on a specific side of a Tru.
    --|
    --| Parameter Description:
    --|   A_Tru is the Tru being acted on.
    --|   A_Tru_Side is the side of the Tru to be updated
    --|   Volts is the Voltage to be given to the Tru
    --|   Load_Conversion is the LCF to be given to the Tru
    --|*****************************************************************
    begin
        A_Tru.Points (A_Tru_Side).V := Volts;
        A_Tru.Points (A_Tru_Side).Lcf := Load_Conversion;
end Give_Voltage_Lcf_To;

pragma Page;
```

```
procedure Give_Current_To (A_Tru : In Tru;
                           A_Tru_Side : In Tru_Side_Names;
                           Load : In Eu.Current) Is
   --|*****************************************************************
   --| Description:
   --|   Places Current on a specific side of a Tru.
   --|
   --| Parameter Description:
   --|   A_Tru is the Tru being acted on.
   --|   A_Tru_Side is the side of the Tru to be updated
   --|   Current is declared in Electrical_Units
   --|*****************************************************************
   begin
      A_Tru.Points (A_Tru_Side).I:= Load;
end Give_Current_To;

pragma Page;
```

```
procedure Give_Power_Info_To (A_Tru : in Tru;
          A_Tru_Side : in Tru_Side_Names;
          External_Power_Info : in Eu.Power_Info) is
    --|*****************************************************************
    --| Description:
    --|   Places Power_Info on a specific side of a Tru.
    --|
    --| Parameter Description:
    --|   A_Tru is the Tru being acted on.
    --|   A_Tru_Side is the side of the Tru to be updated
    --|   Power_Info is declared in Electrical_Units
    --|*****************************************************************
    begin
        A_Tru.Points (A_Tru_Side):= External_Power_Info;
end Give_Power_Info_To;

pragma Page;
```

```
function Get_Power_Info_From (A_Tru : In Tru;
          A_Tru_Side : In Tru_Side_Names)
          return Eu.Power_Info Is
     --|*****************************************************************
     --| Description:
     --|   Returns Power_Info associated with a specific side of an Tru.
     --|
     --| Parameter Description:
     --|   A_Tru is the Tru being acted on.
     --|   A_Tru_Side is the side queried
     --|   Power_Info is declared in Electrical_Units
     --|*****************************************************************
     The_Voltage : Eu.Voltage := Eu.Floating_Voltage;
     The_Current : Eu.Current := Eu.No_Current;
     The_Lcf : Eu.Load_Conversion_Factor := Eu.No_Load_Conversion;
     The_Power_Info : Eu.Power_Info;

     begin
          case A_Tru_Side Is
               when Ac_Side =>
                    The_Current :=
                      A_Tru.Points (Opposite_Side (A_Tru_Side)).I + A_Tru.Load;
               when Dc_Side =>
                    The_Lcf := A_Tru.Lcf;
                    The_Voltage := A_Tru.Points (Opposite_Side (A_Tru_Side)).V;
          end case;

          The_Power_Info.V := The_Voltage;
          The_Power_Info.I := The_Current;
          The_Power_Info.Lcf := The_Lcf;

          RETURN The_Power_Info;
end Get_Power_Info_From;

--|*****************************************************************
--| Modification History:
--|   09Oct87   tac   Reinstated enumeration types Tru_side_names.
--|   05Oct87   tac   Replaced all occurances of "connection" with
--|                   "Point".
--| 30Apr87 kl  Updated read routines to reflect operation of a Tru.
--| 28Apr87 kl  Added power_info routines
--| 20Mar87kl   Created
--|
--|-----------------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*****************************************************************
end Tru_Object_Manager;

pragma Page;
```

D.8 Package Bus_Object_Manager

```
--|*****************************************************************
--| Module Name:
--|   Bus_Object_Manager
--|
--| Module Type:
--|   Package Specification
--|
--| Module Purpose:
--|   This package implements the type manager for objects
--|   which simulate the electrical system Bus.
--|   This management entails creation of Bus objects,
--|   update, maintenance of its state, and state
--|   reporting capabilities.
--|-----------------------------------------------------------------
--| Module Description:
--|   The  Bus_Object_Manager provides a means to create
--|   an Bus object via the New_Bus operation and returns
--|   an identification for the Bus, which is to be used when
--|   updating/accessing the Bus object's state as described below.
--|
--|   The Bus_Object_Manager provides a means of obtaining
--|   state information via the:
--|         1)  Get_Number_Of_Points_From
--|   operation, yielding the following internal state information:
--|         1)  Number_Of_Points: Integer
--|
--|   There are also four functions that are common to every object:
--|
--|  procedure Give_Voltage_Lcf_To (A_Bus : in Bus;
--|              A_Bus_Side : in Bus_Side_Names;
--|              Volts : in EU.Voltage;
--|              Load_Conversion : in EU.Load_Conversion_Factor);
--|
--|  procedure Give_Current_To (A_Bus : in Bus;
--|                                 A_Bus_Side : in Bus_Side_Names;
--|                                 Load : in EU.Current);
--|
--|  procedure Give_Power_Info_To (A_Bus : in Bus;
--|         A_Bus_Side : in Bus_Side_Names;
--|         External_Power_Info : in EU.Power_Info);
--|
--|  function Get_Power_Info_From (A_Bus : in Bus;
--|         A_Bus_Side : in Bus_Side_Names)
--|         return EU.Power_Info;
--|
--| Notes:
--|   Bus is an element of an electrical circuit.  A bus object exists
```

```
--| between every other pair of connected objects.  A bus object may
--| have any number of connections.  The bus object operations implement
--| Kirchoff's current and voltage laws.
--|*************************************************************

with Electrical_Units;

package Bus_Object_Manager is

    package Eu renames Electrical_Units;

    type Bus is private;
    Maximum_Points_On_A_Bus : constant Integer := 50;
    subtype Bus_Side_Names is Integer range 1 .. Maximum_Points_On_A_Bus;

    function New_Bus (Number_Of_Points : in Integer) return Bus;
    --|*********************************************************
    --| Description:
    --|   Creates a new Bus as a private type.
    --|   This function returns a pointer to a new Bus object
    --|   representation.  This pointer will be used to access
    --|   the object for state update and state reporting purposes.
    --|
    --| Parameter Description:
    --|   Number_Of_Points the default state of the Number_Of_Points
    --|   Bus is the access to the private data representaion
    --|
    --|*********************************************************

    procedure Give_Voltage_Lcf_To (A_Bus : in Bus;
                A_Bus_Side : in Bus_Side_Names;
                Volts : in Eu.Voltage;
                Load_Conversion : in Eu.Load_Conversion_Factor);
    --|*********************************************************
    --| Description:
    --|   Places Voltage and LCF on a specific side of a Bus.
    --|
    --| Parameter Description:
    --|   A_Bus is the Bus being acted on.
    --|   A_Bus_Side is the side of the Bus to be updated
    --|   Volts is the Voltage to be given to the Bus
    --|   Load_Conversion is the LCF to be given to the Bus
    --|*********************************************************

    procedure Give_Current_To (A_Bus : in Bus;
                               A_Bus_Side : in Bus_Side_Names;
                               Load : in Eu.Current);
    --|*********************************************************
    --| Description:
    --|   Places Current on a specific side of a Bus.
    --|
    --| Parameter Description:
```

```
    --|  A_Bus is the Bus being acted on.
    --|  A_Bus_Side is the side of the Bus to be updated
    --|  Current is declared in Electrical_Units
    --|*************************************************************

    procedure Give_Power_Info_To (A_Bus : in Bus;
          A_Bus_Side : in Bus_Side_Names;
          External_Power_Info : in Eu.Power_Info);
    --|*************************************************************
    --| Description:
    --|  Places Power_Info on a specific side of a Bus.
    --|
    --| Parameter Description:
    --|  A_Bus is the Bus being acted on.
    --|  A_Bus_Side is the side of the Bus to be updated
    --|  Power_Info is declared in Electrical_Units
    --|*************************************************************

    function Get_Power_Info_From (A_Bus : in Bus;
          A_Bus_Side : in Bus_Side_Names)
          return Eu.Power_Info;
    --|*************************************************************
    --| Description:
    --|  Returns Power_Info associated with a specific side of an Bus.
    --|
    --| Parameter Description:
    --|  A_Bus is the Bus being acted on.
    --|  A_Bus_Side is the side queried
    --|  Power_Info is declared in Electrical_Units
    --|*************************************************************

    function Get_Number_Of_Points_From (A_Bus : in Bus) return Integer;
    --|*************************************************************
    --| Description:
    --|  Returns the state of Number_Of_Points
    --|
    --| Parameter Description:
    --|  A_Bus is the Bus queried
    --|  Integer is the state of the Number_Of_Points
    --|
    --|*************************************************************

pragma Inline (
    New_Bus,
    Give_Voltage_Lcf_To,
    Give_Current_To,
    Give_Power_Info_To,
    Get_Power_Info_From,
    Get_Number_Of_Points_From
    );
```

```
private
    type Bus_Representation (Number_Of_Points:Bus_Side_Names);
    — incomplete type, defined in package body

    type Bus is access Bus_Representation;    — pointer to an Bus representation

—|*****************************************************************
—| Modification History:
—| 28Feb89  kl  changed all occurances of "wire" to "bus"
—|  21Apr87  kl changed connection arguments to wire arguments
—|  28Apr87  kl Updated to include power_info routines
—|  20Mar87  kl Created
—|
—|-----------------------------------------------------------------
—| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
—|*****************************************************************
end Bus_Object_Manager;

pragma Page;
```

D.9 Package Body Bus_Object_Manager

```
--|*****************************************************************
--| Module Name:
--|   Bus_Object_Manager
--|
--| Module Type:
--|   Package Body
--|
--|-----------------------------------------------------------------
--| Module Description:
--|   Manipulates private data structures that represent
--|   a Bus, and contact points on a Bus.
--|
--| Notes:
--|       none
--|*****************************************************************

package body Bus_Object_Manager is

    type Bus_Values is array (Bus_Side_Names range
) of Eu.Power_Info;


    -- representation of a Bus
    --
    type Bus_Representation (Number_Of_Points: Bus_Side_Names)is
        record
            Inputs : Bus_Values (1 .. Number_Of_Points);
            Outputs : Bus_Values (1 .. Number_Of_Points);
            Load_Values_Have_Changed: Boolean := False;
            Voltage_Values_Have_Changed: Boolean:= False;
        end record;

    -- functions required for operations on EU values
    -- without using a USE clause, these functions are not
    -- available, so they have to be explicitly renamed
    --
    function "+" (A_Lcf: In Eu.Load_Conversion_Factor;
                  Another_Lcf: In Eu.Load_Conversion_Factor)
                  return Eu.Load_Conversion_Factor renames Eu."+";
    function "-" (A_Lcf: In Eu.Load_Conversion_Factor;
                  Another_Lcf: In Eu.Load_Conversion_Factor)
                  return Eu.Load_Conversion_Factor renames Eu."-";
    function "/" (A_Lcf: In Eu.Load_Conversion_Factor;
                  Another_Lcf: In Eu.Load_Conversion_Factor)
                  return Eu.Load_Conversion_Factor renames Eu."/";
    function "=" (A_Lcf: In Eu.Load_Conversion_Factor;
                  Another_Lcf: In Eu.Load_Conversion_Factor)
                  return Boolean renames Eu."=";
```

```
    function "=" (A_Current: In Eu.Current;
                  Another_Current: In Eu.Current)
                  return Boolean renames Eu."=";
    function "+" (A_Current: In Eu.Current;
                  Another_Current: In Eu.Current)
                  return Eu.Current renames Eu."+";
    function "-" (A_Current: In Eu.Current;
                  Another_Current: In Eu.Current)
                  return Eu.Current renames Eu."+";
    function "=" (A_Voltage: In Eu.Voltage;
                  Another_Voltage: In Eu.Voltage)
                  return Boolean renames Eu."=";
    function ">" (A_Voltage: In Eu.Voltage;
                  Another_Voltage: In Eu.Voltage)
                  return Boolean renames Eu.">";

pragma Page;
```

```
function New_Bus (Number_Of_Points : In Integer) return Bus is
    --|*****************************************************************
    --| Description:
    --|   Creates a new Bus as a private type.
    --|   This function returns a pointer to a new Bus object
    --|   representation.  This pointer will be used to identify
    --|   the object for state update and state reporting purposes.
    --|
    --| Parameter Description:
    --|   Number_Of_Points the default state of the Number_Of_Points
    --|   Bus is the access to the private data representaion
    --|
    --|*****************************************************************

        The_New_Object: Bus:= new Bus_Representation (Number_Of_Points);

    begin
        RETURN The_New_Object;
end New_Bus;

pragma Page;
```

```
function "*" (A_Current : in Eu.Current;
              A_Lcf : in Eu.Load_Conversion_Factor)
             return Eu.Current is
   --|*****************************************************************
   --| Description:
   --|   Function "*" is used for determining current propagation.
   --|
   --| Parameter Description:
   --|   A_Current and A_Lcf are values that need to be multiplied
   --|   in the determination of a current value for propagation.
   --|
   --|   return value of Current will be propagated to other
   --|   connections.
   --|
   --| Notes:
   --|   none
   --|*****************************************************************

   begin
      RETURN Eu.Current (Float (A_Current) * Float (A_Lcf));
end "*";

pragma Page;
```

```
procedure Give_Voltage_Lcf_To (A_Bus : In Bus;
                A_Bus_Side : In Bus_Side_Names;
                Volts : In Eu.Voltage;
                Load_Conversion : In Eu.Load_Conversion_Factor) Is
    --|**********************************************************************
    --| Description:
    --|  Places Voltage and LCF on a specific side of a Bus.
    --|
    --| Parameter Description:
    --|  A_Bus is the Bus being acted on.
    --|  A_Bus_Side is the side of the Bus to be updated
    --|  Volts is the Voltage to be given to the Bus
    --|  Load_Conversion is the LCF to be given to the Bus
    --|**********************************************************************
    begin
        If A_Bus.Inputs(A_Bus_Side).V /= Volts then
            A_Bus.Voltage_Values_Have_Changed := True;
            A_Bus.Inputs(A_Bus_Side).V := Volts;
        end If;

        If A_Bus.Inputs(A_Bus_Side).Lcf /= Load_Conversion then
            A_Bus.Load_Values_Have_Changed := True;
            A_Bus.Inputs(A_Bus_Side).Lcf := Load_Conversion;
        end If;
end Give_Voltage_Lcf_To;

pragma Page;
```

```
procedure Give_Current_To (A_Bus : In Bus;
                           A_Bus_Side : In Bus_Side_Names;
                           Load : In Eu.Current) Is
   --|*****************************************************************
   --| Description:
   --|   Places Current on a specific side of a Bus.
   --|
   --| Parameter Description:
   --|   A_Bus is the Bus being acted on.
   --|   A_Bus_Side is the side of the Bus to be updated
   --|   Current is declared in Electrical_Units
   --|*****************************************************************
   begin
      If A_Bus.Inputs(A_Bus_Side).I /= Load then
         A_Bus.Load_Values_Have_Changed := True;
         A_Bus.Inputs(A_Bus_Side).I := Load;
      end If;
   end Give_Current_To;

pragma Page;
```

```
procedure Give_Power_Info_To (A_Bus : In Bus;
          A_Bus_Side : In Bus_Side_Names;
          External_Power_Info : In Eu.Power_Info) Is
    --|**************************************************************
    --| Description:
    --|   Places Power_Info on a specific side of a Bus.
    --|
    --| Parameter Description:
    --|   A_Bus is the Bus being acted on.
    --|   A_Bus_Side is the side of the Bus to be updated
    --|   Power_Info is declared in Electrical_Units
    --|**************************************************************
    begin
        If A_Bus.Inputs(A_Bus_Side).V /= External_Power_Info.V then
            A_Bus.Voltage_Values_Have_Changed := True;
            A_Bus.Inputs(A_Bus_Side).V := External_Power_Info.V;
        end If;

        If A_Bus.Inputs(A_Bus_Side).I /= External_Power_Info.I or
         A_Bus.Inputs(A_Bus_Side).Lcf /= External_Power_Info.Lcf then
            A_Bus.Load_Values_Have_Changed := True;
            A_Bus.Inputs(A_Bus_Side).I := External_Power_Info.I;
            A_Bus.Inputs(A_Bus_Side).Lcf := External_Power_Info.Lcf;
        end If;
end Give_Power_Info_To;

pragma Page;
```

```
function Get_Power_Info_From (A_Bus : in Bus;
        A_Bus_Side : in Bus_Side_Names)
        return Eu.Power_Info is
   --|**************************************************************
   --| Description:
   --|  Returns Power_Info associated with a specific side of an Bus.
   --|
   --| Parameter Description:
   --|  A_Bus is the Bus being acted on.
   --|  A_Bus_Side is the side queried
   --|  Power_Info is declared in Electrical_Units
   --|**************************************************************
      The_Voltage: Eu.Voltage;
      The_Lcf : Eu.Load_Conversion_Factor := Eu.No_Load_Conversion;
      The_Current: Eu.Current := Eu.No_Current;

   begin
      If A_Bus.Voltage_Values_Have_Changed then
       for A_Side in A_Bus.Inputs'First .. A_Bus.Inputs'Last loop
          The_Voltage := Eu.Floating_Voltage;

           for A_Point in A_Bus.Inputs'First .. A_Bus.Inputs'Last loop
              If A_Point /= A_Side then
               If A_Bus.Inputs (A_Point).V > The_Voltage then
                  The_Voltage := A_Bus.Inputs (A_Point).V;
               end if;
              end if;
           end loop;

          A_Bus.Outputs(A_Side).V := The_Voltage;
       end loop;

       A_Bus.Voltage_Values_Have_Changed := False;
      end if;

      If A_Bus.Load_Values_Have_Changed then
       for A_Point in A_Bus.Inputs'First .. A_Bus.Inputs'Last loop
          The_Lcf := The_Lcf + A_Bus.Inputs(A_Point).Lcf;
       end loop;

       for A_Side in A_Bus.Outputs'First .. A_Bus.Outputs'Last loop
          The_Current := Eu.No_Current;

          for A_Point in A_Bus.Outputs'First .. A_Bus.Outputs'Last loop
           If (A_Point /= A_Side) and then
              (The_Lcf - A_Bus.Inputs(A_Point).Lcf /= 0.0) then
            The_Current := The_Current + (A_Bus.Inputs(A_Point).I *
               (A_Bus.Inputs(A_Side).Lcf /
                (The_Lcf - A_Bus.Inputs(A_Point).Lcf)));
```

```
            end if;
        end loop;

        A_Bus.Outputs(A_Side).I := The_Current;
        A_Bus.Outputs(A_Side).Lcf := The_Lcf - A_Bus.Inputs(A_Side).Lcf;
      end loop;
      A_Bus.Load_Values_Have_Changed := False;
    end if;

    RETURN A_Bus.Outputs(A_Bus_Side);
end Get_Power_Info_From;

pragma Page;
```

```
function Get_Number_Of_Points_From (A_Bus : In Bus) return Integer is
    —|*****************************************************************
    —| Description:
    —|   Returns the state of Number_Of_Points
    —|
    —| Parameter Description:
    —|   A_Bus is the Bus queried
    —|   integer is the Number_Of_Points
    —|*****************************************************************
    begin
        RETURN A_Bus.Number_Of_Points;
end Get_Number_Of_Points_From;

—|*************************************************************
—| Modification History:
—|  28Feb89   kl    changed all occurances of "wire" with "bus"
—|   09Oct87   tac   Reinstated enumeration types Bus_side_names.
—|   05Oct87   tac   Replaced all occurances of "connection" with
—|              "Point".
—|  30Apr87   kl   Updated read routines to reflect operation of a wire
—|  28Apr87   kl   Added power_info routines
—|  20Mar87   kl   Created
—|
—|-------------------------------------------------------------------------
—| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
—|*************************************************************
end Bus_Object_Manager;
```

D.10 Package Dc_Power_System_Aggregate

```
--|*****************************************************************
--| Module Name:
--|   Dc_Power_System_Aggregate
--|
--| Module Type:
--|   Package Specification
--|
--| Module Purpose:
--|   This package establishes the object aggregates which represent the lowest
--|   level of dc power.
--|
--|-----------------------------------------------------------------
--| Module Description:
--|   The structures which form the aggregate of object information
--|   are defined here. Instantiation of the objects
--|   occurs in a set of constant arrays.
--|
--| Notes:
--|   This package has no body
--|*****************************************************************

--  For the private type CB, the New_Cb operation,
--  and the types of its parameters.
--
with Cb_Object_Manager;

--  For the private type TRU, the New_Tru operation,
--   and the types of its parameters.
--
with Tru_Object_Manager;

--  For the private type Bus, the New_Bus operation,
--   and the types of its parameters.
--
with Bus_Object_Manager;
with Global_Types;

package Dc_Power_System_Aggregate is

    type Cb_Names is (
      -- CB's between TRUs and tie_bus_1
      --
      Cb_1_1,
      Cb_2_1,
      Cb_3_1,

      -- CB between tie_bus_1 and tie_bus_2
      --
      Cb_Tb_1_2,
```

```
— CB between tie_bus_2 and dc_main_4
—
Cb_Tb_2_1,

— CBs tied to dc_main_1
—
Cb_1_2,
Cb_1_3,

— CB tied to dc_main_2
—
Cb_2_2,

— CBs tied to dc_main_3
—
Cb_3_2,
Cb_3_3,
Cb_3_4,

— CB between tie_bus_1 and dc_power_bus_7
—
Cb_Tb_1_1,

— CBs between dc_main_4 and dc_power_bus 8, 9, and 10
Cb_4_1,
Cb_4_2,
Cb_4_3
);

— define a table in which the objects are instantiated and
— can be referenced by the name.
—
Named_Cbs : constant array (Cb_Names) of Cb_Object_Manager.Cb := (
 Cb_1_1 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_2_1 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_3_1 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_Tb_1_2 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_Tb_2_1 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_1_2 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_1_3 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_2_2 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_3_2 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
```

```
 Cb_3_3 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_3_4 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_Tb_1_1 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_4_1 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_4_2 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0),
 Cb_4_3 => Cb_Object_Manager.New_Cb
      (Position => Cb_Object_Manager.Closed, Rating => 50.0)
     );

--
type Tru_Names is (
  Tru_1,
  Tru_2,
  Tru_3,
  Tru_4,
  Tru_5,
  Tru_6);

-- define a table which instantiates TRU objects and allows them
-- to be referenced by name.
--
Named_Trus : constant array (Tru_Names) of Tru_Object_Manager.Tru := (
 Tru_1 => Tru_Object_Manager.New_Tru (Lcf => 1.0, Load => 1.5),
 Tru_2 => Tru_Object_Manager.New_Tru (Lcf => 1.0, Load => 1.5),
 Tru_3 => Tru_Object_Manager.New_Tru (Lcf => 1.0, Load => 1.5),
 Tru_4 => Tru_Object_Manager.New_Tru (Lcf => 1.0, Load => 1.5),
 Tru_5 => Tru_Object_Manager.New_Tru (Lcf => 1.0, Load => 1.5),
 Tru_6 => Tru_Object_Manager.New_Tru (Lcf => 1.0, Load => 1.5));

--
type Bus_Names is (

    -- Buses between TRUs and tie_bus_1
    --
    Dc_Main_1,
    Dc_Main_2,
    Dc_Main_3,

    Tie_Bus_1,

    -- Bus between tie_bus_1 and dc_main_4
    --
    Tie_Bus_2,

    Dc_Main_4,
```

```
   — Dc Power Buses
   —
   Dc_Power_Bus_1,
   Dc_Power_Bus_2,
   Dc_Power_Bus_3,
   Dc_Power_Bus_4,
   Dc_Power_Bus_5,
   Dc_Power_Bus_6,
   Dc_Power_Bus_7,
   Dc_Power_Bus_8,
   Dc_Power_Bus_9,
   Dc_Power_Bus_10
   );

— define a table which instantiates Bus objects and allows them
— to be referenced by name.
—
Named_Buses : constant array (Bus_Names) of Bus_Object_Manager.Bus := (
 Dc_Main_1 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 4),
 Dc_Main_2 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 3),
 Dc_Main_3 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 5),
 Tie_Bus_1 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 5),
 Tie_Bus_2 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 5),
 Dc_Main_4 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 4),
 Dc_Power_Bus_1 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
 Dc_Power_Bus_2 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
 Dc_Power_Bus_3 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
 Dc_Power_Bus_4 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
 Dc_Power_Bus_5 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
 Dc_Power_Bus_6 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
 Dc_Power_Bus_7 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
 Dc_Power_Bus_8 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
 Dc_Power_Bus_9 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
 Dc_Power_Bus_10 =>
        Bus_Object_Manager.New_Bus (Number_Of_Points => 2)
 );
```

```
   -- matches an element kind with an element name and side
   --
   type Element_Point
      (Element : Global_Types.Element_Class := Global_Types.A_Cb) is
      record
         case Element is
            when Global_Types.A_Cb =>
               Cb_Element : Cb_Names;
               Cb_Side : Cb_Object_Manager.Cb_Side_Names;
            when Global_Types.A_Tru =>
               Tru_Element : Tru_Names;
               Tru_Side : Tru_Object_Manager.Tru_Side_Names;
            when Global_Types.A_Bus =>
               Bus_Element : Bus_Names;
               Bus_Side : Bus_Object_Manager.Bus_Side_Names;
         end case;
      end record;

--|*****************************************************************
--| Modification History:
--| 01Mar89  kl   changed all instances of "wire" to "bus";
--|               removed contractor specific names of components
--| 22Jun88  der  removed _Input_Point and included the
--|               point information in the Bus side.
--| 16Jun88  der  added Input_Point to the Element point record
--|               definition to support the Bus object.
--| 14Jun88  der  Buses added to dc_power_aggregate definition.
--| 09Oct87  tac  Eliminated Enumeration type Points in all
--|               aggregates, and combined arrays which used it.
--|               Reinstated enumeration types tru_side_names, and
--|               cb_side_names. Made changes to support modification to
--|               element_class.
--| 01Sep87  tac  removed packaging around dc_power_node_names,
--|               dc_power_cb_names, dc_power_tru_names; deleted tru_no_2_cb,
--|               tru_no_4_cb, and tru_no_8_cb from list of cb_names enumerated
--|               type; deleted tru_2_node, tru_4_node, and tru_8_node from
--|               node_names enumerated type; adjusted cb_* connection names
--|               after deleting cb_1..cb_6 from connections enumerated type;
--|               made changes to the_node_connections because of type
--|               connections change; deleted entries for cb_1..cb_6 in
--|               the_connections constant array; added type side_names
--|               from node_object_manager package; removed node from type
--|               element_class; removed node case option from type
--|               connection_between_elements
--| 21Jul87  kl   renamed package to dc_power_aggregate;
--|               added dc_power_node_names, dc_power_cb_names, and
--|               dc_power_tru_names as subpackages
--| 04May87  kl   removed class node; they are redundant
--| 03May87  kl   removed class terminus; they can be modeled with
--|               node connections
--| 02May87  kl   updated element names used in
--|               the_node_connections and the_connections.
```

```
—| 24Apr87  kl   created with element classes: cb, tru, terminus,
—|               and node
—|-------------------------------------------------------------------------
—| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
—|*************************************************************
end Dc_Power_System_Aggregate;

pragma Page;
```

D.11 Package Dummy_System_Aggregate

```
--|*****************************************************************
--| Module Name:
--|   Dummy_System_Aggregate
--|
--| Module Type:
--|   Package Specification
--|
--| Module Purpose:
--|   This package establishes the object aggregates which represent the lowest
--|   level of a dummy system.
--|
--|-----------------------------------------------------------------
--| Module Description:
--|   The structures which form the aggregate of object information
--|   are defined here. Instantiation of the objects
--|   occurs in a set of constant arrays.
--|
--| Notes:
--|   This package has no body
--|*****************************************************************
--  For the private type Cb, and the New_Cb operation and
--  the types of its parameters.
--
with Cb_Object_Manager;
with Global_Types;

package Dummy_System_Aggregate is

    -- Dummy Cb names
    --
    type Cb_Names is (
        Dummy_Cb_1,
        Dummy_Cb_2,
        Dummy_Cb_3,
        Dummy_Cb_4,
        Dummy_Cb_5,
        Dummy_Cb_6,
        Dummy_Cb_7,
        Dummy_Cb_8,
        Dummy_Cb_9,
        Dummy_Cb_10);

    -- instantiate Cbs for dummy system
    --
    Named_Cbs : constant array (Cb_Names) of Cb_Object_Manager.Cb := (
      Dummy_Cb_1 => Cb_Object_Manager.New_Cb
          (Position => Cb_Object_Manager.Closed, Rating => 50.0),
```

```
    Dummy_Cb_2 => Cb_Object_Manager.New_Cb
        (Position => Cb_Object_Manager.Closed, Rating => 50.0),
    Dummy_Cb_3 => Cb_Object_Manager.New_Cb
        (Position => Cb_Object_Manager.Closed, Rating => 50.0),
    Dummy_Cb_4 => Cb_Object_Manager.New_Cb
        (Position => Cb_Object_Manager.Closed, Rating => 50.0),
    Dummy_Cb_5 => Cb_Object_Manager.New_Cb
        (Position => Cb_Object_Manager.Closed, Rating => 50.0),
    Dummy_Cb_6 => Cb_Object_Manager.New_Cb
        (Position => Cb_Object_Manager.Closed, Rating => 50.0),
    Dummy_Cb_7 => Cb_Object_Manager.New_Cb
        (Position => Cb_Object_Manager.Closed, Rating => 50.0),
    Dummy_Cb_8 => Cb_Object_Manager.New_Cb
        (Position => Cb_Object_Manager.Closed, Rating => 50.0),
    Dummy_Cb_9 => Cb_Object_Manager.New_Cb
        (Position => Cb_Object_Manager.Closed, Rating => 50.0),
    Dummy_Cb_10 => Cb_Object_Manager.New_Cb
        (Position => Cb_Object_Manager.Closed, Rating => 50.0));

  -- matches an element kind with an element name and side
  --
  type Element_Point
      (Element : Global_Types.Element_Class := Global_Types.A_Cb) is
      record
          case Element is
              when Global_Types.A_Cb =>
                  Cb_Element : Cb_Names;
                  Cb_Side : Cb_Object_Manager.Cb_Side_Names;
              when others =>
                  null;
          end case;
      end record;

--|*****************************************************************
--| Modification History:
--|  03Nov88  kl   Changed objects from wires to CBs
--|  22Jun88  der  Deleted cbs, and removed Input_Point
--|                from the element point type.
--|  16Jun88  der  Added wires to the dummy aggregate.
--|                Added Input_Point to the Element point record
--|                definition to support the wire object
--|  09Oct87  tac  Eliminated Enumeration type Points in all
--|                aggregates, and combined arrays which used it.
--|  25sep87  tac  created
--|---------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*****************************************************************
end Dummy_System_Aggregate;
pragma Page;
```

D.12 Package Ac_Power_System_Aggregate

```
--|*****************************************************************
--| Module Name:
--|   Ac_Power_System_Aggregate
--|
--| Module Type:
--|   Package Specification
--|
--| Module Purpose:
--|   This package establishes the object aggregates which represent the lowest
--|   level of AC power.
--|
--|-----------------------------------------------------------------
--| Module Description:
--|   The structures which form the aggregate of object information
--|   are defined here. Instantiation of the objects themselves
--|   occur in a set of constant arrays.
--|
--| Notes:
--|   This package has no body
--|*****************************************************************
--  For the private type Bus, the New_Bus operation,
--  and the types of its parameters.
--
with Bus_Object_Manager;
with Global_Types;

package Ac_Power_System_Aggregate is

    -- Buses which provide ac source to dc power
    --
    type Bus_Names is (
        Ac_Bus_1,
        Ac_Bus_2,
        Ac_Bus_3,
        Ac_Bus_4,
        Ac_Bus_5,
        Ac_Bus_6
        );

    -- Instantiate the Bus objects and collect them so they can
    -- be referenced by Bus name.
    --
    Named_Buses : constant array (Bus_Names) of Bus_Object_Manager.Bus := (
      Ac_Bus_1 => Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
        Ac_Bus_2 => Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
        Ac_Bus_3 => Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
        Ac_Bus_4 => Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
```

```
        Ac_Bus_5 => Bus_Object_Manager.New_Bus (Number_Of_Points => 2),
        Ac_Bus_6 => Bus_Object_Manager.New_Bus (Number_Of_Points => 2)
        );

    -- matches an element with its element name and side.
    --
    type Element_Point
        (Element : Global_Types.Element_Class := Global_Types.A_Bus) is
        record
            case Element is

                when Global_Types.A_Bus =>
                    Bus_Element : Bus_Names;
                    Bus_Side : Bus_Object_Manager.Bus_Side_Names;
                when others =>
                    null;
            end case;
        end record;

--|*****************************************************************
--| Modification History:
--| 01Mar89  kl   changed all occurrences of "wire" to "bus"
--|  22Jun88  der  deleted Cbs from ac_power_aggregate.
--|  14Jun88  der  added wires to ac_power_aggregate definition
--|  09Oct87  tac  Eliminated Enumeration type Points in all
--|               aggregates, and combined arrays which used it.
--|  25sep87  tac  Created
--|
--|-----------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*****************************************************************
end Ac_Power_System_Aggregate;

pragma Page;
```

D.13 Package Cb_Linkage_Interface

```
--|*****************************************************************
--| Module Name:
--|   Cb_Linkage_interface
--|
--| Module Type:
--|   Package Specification
--|
--| Module Purpose:
--|   This package provides scaffolding for testing Dc_Power
--|-----------------------------------------------------------------
--| Module Description:
--|   This package provides names for CBs in the software representation of the
--|   linkage buffer.  When CBs are updated relative to the linkage
--|   buffer, these names would be used to determine the current
--|   value of the a CB position.
--|
--| The names are chosen to match the CB names used in dc_power.
--|
--| Notes:
--|   none
--|*****************************************************************
with Cb_Object_Manager;
with Dc_Power_System_Aggregate;

package Cb_Linkage_Interface is

    package Dcpa renames Dc_Power_System_Aggregate;

    type Cb_Linkage_Ids is (
      Link_For_Cb_1_1,
      Link_For_Cb_2_1,
      Link_For_Cb_3_1,
      Link_For_Cb_Tb_1_2,
      Link_For_Cb_Tb_2_1,
      Link_For_Cb_1_2,
      Link_For_Cb_1_3,
      Link_For_Cb_2_2,
      Link_For_Cb_3_2,
      Link_For_Cb_3_3,
      Link_For_Cb_3_4,
      Link_For_Cb_Tb_1_1,
      Link_For_Cb_4_1,
      Link_For_Cb_4_2,
      Link_For_Cb_4_3
        );
```

```
   function Get_Hardware_Cb_Position (Hardware_Link : in Cb_Linkage_Ids)
                                        return Cb_Object_Manager.Cb_Position;
   --|*****************************************************************
   --| Description:
   --|   Returns the current value of the linkage buffer CB position.
   --|
   --| Parameter Description:
   --|   Hardware_Link is the name of CB linkage
   --|   Returns current state of CB position from the linkage buffer
   --|*****************************************************************

   procedure Set_Hardware_Cb_Position (
       Hardware_Link : in Cb_Linkage_Ids;
       A_Position: in Cb_Object_Manager.Cb_Position);
   --|*****************************************************************
   --| Description:
   --|   Sets the current value of the linkage buffer CB position.
   --|
   --| Parameter Description:
   --|   Hardware_Link is the name of CB linkage
   --|   A_Position is a new state of CB position
   --|*****************************************************************

   function Get_A_Dc_Cb_Link (A_Dc_Cb: in Dcpa.Cb_Names) return Cb_Linkage_Ids;
   --|*****************************************************************
   --| Description:
   --|   Gets the linkage ID for a Dc Power CB
   --|
   --| Parameter Description:
   --|   A_Dc_Cb is the Dc Power CB
   --|   Returns a CB linkage ID
   --|*****************************************************************

--pragma inline(Get_Hardware_Cb_Position,
--              Set_Hardware_Cb_Position,
--              Get_A_Dc_Cb_Link);

--|*****************************************************************
--| Modification History:
--|   14Nov88   kl   added procedure Set_Hardware_Cb_Position
--|   12Mar87   kl   created
--|
--|-----------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*****************************************************************
end Cb_Linkage_Interface;

pragma Page;
```

D.14 Package Body Cb_Linkage_Interface

```
--|*****************************************************************
--| Module Name:
--|   Cb_Linkage_interface
--|
--| Module Type:
--|   Package Body
--|
--|-----------------------------------------------------------------
--| Module Description:
--|   This package provides scaffolding for testing Dc_Power
--|
--| Notes:
--|   none
--|*****************************************************************

package body Cb_Linkage_Interface is

  -- Define tables for cross referencing Cb linkages
  --
  Dc_Power_Cb_Linkage_Ids : constant array (Dcpa.Cb_Names) of
              Cb_Linkage_Ids := (
       Dcpa.Cb_1_1 => Link_For_Cb_1_1,
       Dcpa.Cb_2_1 => Link_For_Cb_2_1,
       Dcpa.Cb_3_1 => Link_For_Cb_3_1,
       Dcpa.Cb_Tb_1_2 => Link_For_Cb_Tb_1_2,
       Dcpa.Cb_Tb_2_1 => Link_For_Cb_Tb_2_1,
       Dcpa.Cb_1_2 => Link_For_Cb_1_2,
       Dcpa.Cb_1_3 => Link_For_Cb_1_3,
       Dcpa.Cb_2_2 => Link_For_Cb_2_2,
       Dcpa.Cb_3_2 => Link_For_Cb_3_2,
       Dcpa.Cb_3_3 => Link_For_Cb_3_3,
       Dcpa.Cb_3_4 => Link_For_Cb_3_4,
       Dcpa.Cb_Tb_1_1 => Link_For_Cb_Tb_1_1,
       Dcpa.Cb_4_1 => Link_For_Cb_4_1,
       Dcpa.Cb_4_2 => Link_For_Cb_4_2,
       Dcpa.Cb_4_3 => Link_For_Cb_4_3
          );

     type Cb_Linkage is array (Cb_Linkage_Ids) of
          Cb_Object_Manager.Cb_Position;

     The_Cb_Linkage: Cb_Linkage:= (others => Cb_Object_Manager.Closed);

     function Get_Hardware_Cb_Position (Hardware_Link : in Cb_Linkage_Ids)
                                        return Cb_Object_Manager.Cb_Position is
     --|*****************************************************************
     --| Description:
```

```
    —| Returns the current value of the linkage buffer CB position.
    —|
    —| Parameter Description:
    —| Hardware_Link is the name of CB linkage
    —| Returns current state of CB position from the linkage buffer
    —|
    —| Notes:
    —| none
    —|*****************************************************************
    begin
        RETURN The_Cb_Linkage(Hardware_Link);
    end Get_Hardware_Cb_Position;

pragma Page;
```

```
    procedure Set_Hardware_Cb_Position (Hardware_Link : in Cb_Linkage_Ids;
        A_Position: in Cb_Object_Manager.Cb_Position) is
    --|*****************************************************************
    --| Description:
    --|   Sets the current value of the linkage buffer CB position.
    --|
    --| Parameter Description:
    --|   Hardware_Link is the name of CB linkage
    --|   A_Position is a new state of CB position
    --|
    --| Notes:
    --|   none
    --|*****************************************************************
    begin
        The_Cb_Linkage(Hardware_Link) := A_Position;
    end Set_Hardware_Cb_Position;

pragma Page;
```

```
    function Get_A_Dc_Cb_Link (A_Dc_Cb: In Dcpa.Cb_Names)
        return Cb_Linkage_Ids Is
    --|*****************************************************************
    --| Description:
    --|   Gets the linkage ID for a Dc Power CB
    --|
    --| Parameter Description:
    --|   A_Dc_Cb is the Dc Power CB
    --|   Returns a CB linkage ID
    --|
    --| Notes:
    --|   none
    --|*****************************************************************
    begin
        RETURN Dc_Power_Cb_Linkage_Ids(A_Dc_Cb);
    end Get_A_Dc_Cb_Link;

--|*********************************************************************
--| Modification History:
--|   14Nov88   kl   added procedure set_hardware_cb_position
--|   11Apr87   kl   created
--|
--|---------------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*********************************************************************
end Cb_Linkage_Interface;

pragma Page;
```

D.15 Package Dc_Power_System

```
--|*****************************************************************
--| Module Name:
--|   Dc_Power_System;
--|
--| Module Type:
--|   Package Specification
--|
--| Module Purpose:
--|   This package contains the top-level procedure call to update the
--|   simulation of DC power generation.  It is the sole
--|   interface to DC power from the perspective of the flight executive.
--|-----------------------------------------------------------------
--| Module Description:
--|   This package exports a procedure call, Update_DC_Power.
--|   Update_Dc_Power abstracts all processing necessary to
--|   update the DC power system.  During the call all connections
--|   between objects in DC power are processed. Since the objects
--|   are already at the lowest level, there is no further call to
--|   any nested level. Prior to issuing the Update_Dc_Power call
--|   all outside affects on DC power objects should have been placed on the
--|   objects by the flight executive.
--|
--| Notes:
--|   none
--|*****************************************************************

package Dc_Power_System is

    procedure Update_Dc_Power_System;
    --|*************************************************************
    --|
    --| Description:
    --|   This procedure causes all connections within DC power to be processed.
    --|   It establishes the order in which connections are processed,
    --|   and how many times they are processed.
    --|
    --| Parameter Description:
    --|   None
    --|
    --|*************************************************************

--pragma inline (Update_Dc_Power_System);

--|*****************************************************************
--| Modification History:
--|
--|   20Feb87   mmr   initial version
```

```
--|-------------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*******************************************************************
end Dc_Power_System;

pragma Page;
```

D.16 Package Body Dc_Power_System

```
--|*****************************************************************
--| Module Name:
--|   Dc_Power_System
--|
--| Module Type:
--|   Package Body
--|
--|-----------------------------------------------------------------
--| Module Description:
--|   This package containes a set of subprograms to process
--|   connections between DC power objects.  Subprograms have been
--|   defined to process connections with respect to voltage and Lcf,
--|   and load. A single procedure, Update_Dc_Power, abstracts
--|   updating of DC power objects which have already had outside
--|   effects put on them by the flight executive.
--|
--| Notes:
--|   none
--|*****************************************************************
with Global_Types;
with Electrical_Units;

with Cb_Object_Manager;
with Tru_Object_Manager;
with Bus_Object_Manager;

with Dc_Power_System_Aggregate;

package body Dc_Power_System is

  package Dcpa renames Dc_Power_System_Aggregate;

  type Dc_Power_System_Connection_Names is (
     -- Connections from TRUs to dc_mains 1, 2, and 3
     --
     Connection_7, Connection_8, Connection_9,

     -- Connections from Dc_Main_1 to power bus CBs
     --
     Connection_10, Connection_11,

     -- Connection from Dc_Main_2 to power bus CB
     --
     Connection_12,
```

```
   — Connections from Dc_Main_3 to power bus CBs
   —
   Connection_13, Connection_14, Connection_15,

   — Connections from dc_mains 1, 2, and 3 to tie_bus_1 CBs
   —
   Connection_16, Connection_17, Connection_18,

   — Connections from tie_bus_1 CBs to Tie_Bus_1
   —
   Connection_19, Connection_20, Connection_21, Connection_22, Connection_23,

   — Connections to Tie_Bus_2
   —
   Connection_24, Connection_25, Connection_26, Connection_27, Connection_28,

   — Connections to the Dc_Main_4
   —
   Connection_29, Connection_30, Connection_31, Connection_32,

    — Connections to DC power buses
    —
   Connection_33, Connection_34, Connection_35, Connection_36, Connection_37,
   Connection_38, Connection_39, Connection_40, Connection_41, Connection_42
   );

subtype The_Tie_Bus_1_Connections is Dc_Power_System_Connection_Names
   range Connection_10..Connection_23;

subtype The_Tie_Bus_2_Connections is Dc_Power_System_Connection_Names
   range Connection_24..Connection_28;

— structure for each Connection
—
type A_Dc_Power_System_Connection is array (Integer range 1 .. 2) of
                  Dcpa.Element_Point;

type Dc_Power_System_Connections is array (Dc_Power_System_Connection_Names) of
     A_Dc_Power_System_Connection;

pragma Page;
```

```
-- table which provides a cross reference between each Connection
-- and information on each Point on it.
--
The_Dc_Power_System_Connections : constant Dc_Power_System_Connections := (

   Connection_7 => (
            2 => (Element => Global_Types.A_Bus,
                  Bus_Element => Dcpa.Dc_Main_1,
                  Bus_Side => 1),
            1 => (Element => Global_Types.A_Tru,
                  Tru_Element => Dcpa.Tru_1,
                  Tru_Side => Tru_Object_Manager.Dc_Side)),

   Connection_8 => (
            2 => (Element => Global_Types.A_Bus,
                  Bus_Element => Dcpa.Dc_Main_2,
                  Bus_Side => 1),
            1 => (Element => Global_Types.A_Tru,
                  Tru_Element => Dcpa.Tru_2,
                  Tru_Side => Tru_Object_Manager.Dc_Side)),

   Connection_9 => (
            2 => (Element => Global_Types.A_Bus,
                  Bus_Element => Dcpa.Dc_Main_3,
                  Bus_Side => 1),
            1 => (Element => Global_Types.A_Tru,
                  Tru_Element => Dcpa.Tru_3,
                  Tru_Side => Tru_Object_Manager.Dc_Side)),

   Connection_10 => (
            1 => (Element => Global_Types.A_Bus,
                  Bus_Element => Dcpa.Dc_Main_1,
                  Bus_Side => 2),
            2 => (Element => Global_Types.A_Cb,
                  Cb_Element => Dcpa.Cb_1_2,
                  Cb_Side => Cb_Object_Manager.Side_1)),

   Connection_11 => (
            1 => (Element => Global_Types.A_Bus,
                  Bus_Element => Dcpa.Dc_Main_1,
                  Bus_Side => 3),
            2 => (Element => Global_Types.A_Cb,
                  Cb_Element => Dcpa.Cb_1_3,
                  Cb_Side => Cb_Object_Manager.Side_1)),

   Connection_12 => (
            1 => (Element => Global_Types.A_Bus,
                  Bus_Element => Dcpa.Dc_Main_2,
                  Bus_Side => 2),
            2 => (Element => Global_Types.A_Cb,
```

```
            Cb_Element => Dcpa.Cb_2_2,
            Cb_Side => Cb_Object_Manager.Side_1)),

Connection_13 => (
        1 => (Element => Global_Types.A_Bus,
            Bus_Element => Dcpa.Dc_Main_3,
            Bus_Side => 2),
        2 => (Element => Global_Types.A_Cb,
            Cb_Element => Dcpa.Cb_3_2,
            Cb_Side => Cb_Object_Manager.Side_1)),

Connection_14 => (
        1 => (Element => Global_Types.A_Bus,
            Bus_Element => Dcpa.Dc_Main_3,
            Bus_Side => 3),
        2 => (Element => Global_Types.A_Cb,
            Cb_Element => Dcpa.Cb_3_3,
            Cb_Side => Cb_Object_Manager.Side_1)),

Connection_15 => (
        1 => (Element => Global_Types.A_Bus,
            Bus_Element => Dcpa.Dc_Main_3,
            Bus_Side => 4),
        2 => (Element => Global_Types.A_Cb,
            Cb_Element => Dcpa.Cb_3_4,
            Cb_Side => Cb_Object_Manager.Side_1)),

Connection_16 => (
        1 => (Element => Global_Types.A_Bus,
            Bus_Element => Dcpa.Dc_Main_1,
            Bus_Side => 4),
        2 => (Element => Global_Types.A_Cb,
            Cb_Element => Dcpa.Cb_1_1,
            Cb_Side => Cb_Object_Manager.Side_1)),

Connection_17 => (
        1 => (Element => Global_Types.A_Bus,
            Bus_Element => Dcpa.Dc_Main_2,
            Bus_Side => 3),
        2 => (Element => Global_Types.A_Cb,
            Cb_Element => Dcpa.Cb_2_1,
            Cb_Side => Cb_Object_Manager.Side_1)),

Connection_18 => (
        1 => (Element => Global_Types.A_Bus,
            Bus_Element => Dcpa.Dc_Main_3,
            Bus_Side => 5),
        2 => (Element => Global_Types.A_Cb,
            Cb_Element => Dcpa.Cb_3_1,
            Cb_Side => Cb_Object_Manager.Side_1)),
```

```
Connection_19 => (
        2 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Tie_Bus_1,
           Bus_Side => 1),
        1 => (Element => Global_Types.A_Cb,
           Cb_Element => Dcpa.Cb_1_1,
           Cb_Side => Cb_Object_Manager.Side_2)),

Connection_20 => (
        2 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Tie_Bus_1,
           Bus_Side => 2),
        1 => (Element => Global_Types.A_Cb,
           Cb_Element => Dcpa.Cb_2_1,
           Cb_Side => Cb_Object_Manager.Side_2)),

Connection_21 => (
        2 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Tie_Bus_1,
           Bus_Side => 3),
        1 => (Element => Global_Types.A_Cb,
           Cb_Element => Dcpa.Cb_3_1,
           Cb_Side => Cb_Object_Manager.Side_2)),

Connection_22 => (
        1 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Tie_Bus_1,
           Bus_Side => 4),
        2 => (Element => Global_Types.A_Cb,
           Cb_Element => Dcpa.Cb_Tb_1_1,
           Cb_Side => Cb_Object_Manager.Side_1)),

Connection_23 => (
        1 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Tie_Bus_1,
           Bus_Side => 5),
        2 => (Element => Global_Types.A_Cb,
           Cb_Element => Dcpa.Cb_Tb_1_2,
           Cb_Side => Cb_Object_Manager.Side_1)),

Connection_24 => (
        2 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Tie_Bus_2,
           Bus_Side => 2),
        1 => (Element => Global_Types.A_Tru,
           Tru_Element => Dcpa.Tru_4,
           Tru_Side => Tru_Object_Manager.Dc_Side)),

Connection_25 => (
        2 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Tie_Bus_2,
           Bus_Side => 3),
```

```
            1 => (Element => Global_Types.A_Tru,
               Tru_Element => Dcpa.Tru_5,
               Tru_Side => Tru_Object_Manager.Dc_Side)),

      Connection_26 => (
            2 => (Element => Global_Types.A_Bus,
               Bus_Element => Dcpa.Tie_Bus_2,
               Bus_Side => 4),
            1 => (Element => Global_Types.A_Tru,
               Tru_Element => Dcpa.Tru_6,
               Tru_Side => Tru_Object_Manager.Dc_Side)),

      Connection_27 => (
            2 => (Element => Global_Types.A_Bus,
               Bus_Element => Dcpa.Tie_Bus_2,
               Bus_Side => 1),
            1 => (Element => Global_Types.A_Cb,
               Cb_Element => Dcpa.Cb_Tb_1_2,
               Cb_Side => Cb_Object_Manager.Side_2)),

      Connection_28 => (
            1 => (Element => Global_Types.A_Bus,
               Bus_Element => Dcpa.Tie_Bus_2,
               Bus_Side => 5),
            2 => (Element => Global_Types.A_Cb,
               Cb_Element => Dcpa.Cb_Tb_2_1,
               Cb_Side => Cb_Object_Manager.Side_1)),

      Connection_29 => (
            2 => (Element => Global_Types.A_Bus,
               Bus_Element => Dcpa.Dc_Main_4,
               Bus_Side => 1),
            1 => (Element => Global_Types.A_Cb,
               Cb_Element => Dcpa.Cb_Tb_2_1,
               Cb_Side => Cb_Object_Manager.Side_2)),

      Connection_30 => (
            1 => (Element => Global_Types.A_Bus,
               Bus_Element => Dcpa.Dc_Main_4,
               Bus_Side => 2),
            2 => (Element => Global_Types.A_Cb,
               Cb_Element => Dcpa.Cb_4_1,
               Cb_Side => Cb_Object_Manager.Side_1)),

      Connection_31 => (
            1 => (Element => Global_Types.A_Bus,
               Bus_Element => Dcpa.Dc_Main_4,
               Bus_Side => 3),
            2 => (Element => Global_Types.A_Cb,
               Cb_Element => Dcpa.Cb_4_2,
               Cb_Side => Cb_Object_Manager.Side_1)),
```

```
Connection_32 => (
        1 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Dc_Main_4,
           Bus_Side => 4),
        2 => (Element => Global_Types.A_Cb,
           Cb_Element => Dcpa.Cb_4_3,
           Cb_Side => Cb_Object_Manager.Side_1)),

Connection_33 => (
        2 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Dc_Power_Bus_1,
           Bus_Side => 1),
        1 => (Element => Global_Types.A_Cb,
           Cb_Element => Dcpa.Cb_1_2,
           Cb_Side => Cb_Object_Manager.Side_2)),

Connection_34 => (
        2 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Dc_Power_Bus_2,
           Bus_Side => 1),
        1 => (Element => Global_Types.A_Cb,
           Cb_Element => Dcpa.Cb_1_3,
           Cb_Side => Cb_Object_Manager.Side_2)),

Connection_35 => (
        2 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Dc_Power_Bus_3,
           Bus_Side => 1),
        1 => (Element => Global_Types.A_Cb,
           Cb_Element => Dcpa.Cb_2_2,
           Cb_Side => Cb_Object_Manager.Side_2)),

Connection_36 => (
        2 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Dc_Power_Bus_4,
           Bus_Side => 1),
        1 => (Element => Global_Types.A_Cb,
           Cb_Element => Dcpa.Cb_3_2,
           Cb_Side => Cb_Object_Manager.Side_2)),

Connection_37 => (
        2 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Dc_Power_Bus_5,
           Bus_Side => 1),
        1 => (Element => Global_Types.A_Cb,
           Cb_Element => Dcpa.Cb_3_3,
           Cb_Side => Cb_Object_Manager.Side_2)),

Connection_38 => (
        2 => (Element => Global_Types.A_Bus,
           Bus_Element => Dcpa.Dc_Power_Bus_6,
           Bus_Side => 1),
```

```
            1 => (Element => Global_Types.A_Cb,
               Cb_Element => Dcpa.Cb_3_4,
               Cb_Side => Cb_Object_Manager.Side_2)),

      Connection_39 => (
            2 => (Element => Global_Types.A_Bus,
               Bus_Element => Dcpa.Dc_Power_Bus_7,
               Bus_Side => 1),
            1 => (Element => Global_Types.A_Cb,
               Cb_Element => Dcpa.Cb_Tb_1_1,
               Cb_Side => Cb_Object_Manager.Side_2)),

      Connection_40 => (
            2 => (Element => Global_Types.A_Bus,
               Bus_Element => Dcpa.Dc_Power_Bus_8,
               Bus_Side => 1),
            1 => (Element => Global_Types.A_Cb,
               Cb_Element => Dcpa.Cb_4_1,
               Cb_Side => Cb_Object_Manager.Side_2)),

      Connection_41 => (
            2 => (Element => Global_Types.A_Bus,
               Bus_Element => Dcpa.Dc_Power_Bus_9,
               Bus_Side => 1),
            1 => (Element => Global_Types.A_Cb,
               Cb_Element => Dcpa.Cb_4_2,
               Cb_Side => Cb_Object_Manager.Side_2)),

      Connection_42 => (
            2 => (Element => Global_Types.A_Bus,
               Bus_Element => Dcpa.Dc_Power_Bus_10,
               Bus_Side => 1),
            1 => (Element => Global_Types.A_Cb,
               Cb_Element => Dcpa.Cb_4_3,
               Cb_Side => Cb_Object_Manager.Side_2))

);

 pragma Page;
```

```
procedure Process_Voltage_Lcf_For_Connection
        (This_Connection : Dc_Power_System_Connection_Names) is
   --|**********************************************************
   --| Description:
   --|  This procedure processes the voltage and Lcf state
   --|  variables for the specified connection.
   --|
   --| Parameter Description:
   --|  This Connection is the Connection to be updated
   --|
   --| Notes:
   --|  none
   --|**********************************************************
   The_Power_Info : Electrical_Units.Power_Info;

begin
   declare
    The_Connection : A_Dc_Power_System_Connection
       renames The_Dc_Power_System_Connections (This_Connection);
   begin
    --
    -- handle a dc power Point
    -- obtain power information from dc power object
    --
    case The_Connection (1).Element is
       when Global_Types.A_Cb =>
        The_Power_Info :=
            Cb_Object_Manager.Get_Power_Info_From (
             A_Cb => Dcpa.Named_Cbs (The_Connection (1).Cb_Element),
             A_Cb_Side => The_Connection (1).Cb_Side);

       when Global_Types.A_Tru =>
        The_Power_Info :=
            Tru_Object_Manager.Get_Power_Info_From (
             A_Tru => Dcpa.Named_Trus (The_Connection (1).Tru_Element),
             A_Tru_Side => The_Connection (1).Tru_Side);

       when Global_Types.A_Bus =>
        The_Power_Info :=
            Bus_Object_Manager.Get_Power_Info_From (
             A_Bus => Dcpa.Named_Buses (The_Connection (1).Bus_Element),
             A_Bus_Side => The_Connection (1).Bus_Side);
    end case;

    -- restore new Voltage and Lcf states to an object
    --
    case The_Connection (2).Element is
       when Global_Types.A_Cb =>
        Cb_Object_Manager.Give_Voltage_Lcf_To (
            A_Cb => Dcpa.Named_Cbs (The_Connection (2).Cb_Element),
```

```
            A_Cb_Side => The_Connection (2).Cb_Side,
            Volts => The_Power_Info.V,
            Load_Conversion => The_Power_Info.Lcf);

        when Global_Types.A_Tru =>
          Tru_Object_Manager.Give_Voltage_Lcf_To (
            A_Tru => Dcpa.Named_Trus (The_Connection (2).Tru_Element),
            A_Tru_Side => The_Connection (2).Tru_Side,
            Volts => The_Power_Info.V,
            Load_Conversion => The_Power_Info.Lcf);

      when Global_Types.A_Bus =>
          Bus_Object_Manager.Give_Voltage_Lcf_To (
            A_Bus => Dcpa.Named_Buses (The_Connection (2).Bus_Element),
            A_Bus_Side => The_Connection (2).Bus_Side,
            Volts => The_Power_Info.V,
            Load_Conversion => The_Power_Info.Lcf);
      end case;
    end;
end Process_Voltage_Lcf_For_Connection;

pragma Page;
```

```
procedure Process_Load_For_Connection
        (This_Connection : Dc_Power_System_Connection_Names) is
   --|*****************************************************************
   --| Description:
   --|  This procedure processes the current for the specified connection.
   --|
   --| Parameter Description:
   --|  This Connection is the Connection to be processed.
   --|
   --| Notes:
   --|  none
   --|*****************************************************************
   The_Power_Info : Electrical_Units.Power_Info;

begin
   declare
    The_Connection : A_Dc_Power_System_Connection
       renames The_Dc_Power_System_Connections (This_Connection);
   begin
    -- obtain the power information from an Dc Power object.
    --
    case The_Connection (2).Element is
       when Global_Types.A_Cb =>
        The_Power_Info :=
            Cb_Object_Manager.Get_Power_Info_From (
             A_Cb => Dcpa.Named_Cbs (The_Connection (2).Cb_Element),
             A_Cb_Side => The_Connection (2).Cb_Side);

       when Global_Types.A_Tru =>
        The_Power_Info :=
            Tru_Object_Manager.Get_Power_Info_From (
             A_Tru => Dcpa.Named_Trus (The_Connection (2).Tru_Element),
             A_Tru_Side => The_Connection (2).Tru_Side);

       when Global_Types.A_Bus =>
        The_Power_Info :=
            Bus_Object_Manager.Get_Power_Info_From (
             A_Bus => Dcpa.Named_Buses (The_Connection (2).Bus_Element),
             A_Bus_Side => The_Connection (2).Bus_Side);
    end case;

    -- restore new load state to the Dc Power object
    --
    case The_Connection (1).Element is
       when Global_Types.A_Cb =>
        Cb_Object_Manager.Give_Current_To (
            A_Cb => Dcpa.Named_Cbs (The_Connection (1).Cb_Element),
            A_Cb_Side => The_Connection (1).Cb_Side,
            Load => The_Power_Info.I);
```

```
            when Global_Types.A_Tru =>
              Tru_Object_Manager.Give_Current_To (
                   A_Tru => Dcpa.Named_Trus (The_Connection (1).Tru_Element),
                   A_Tru_Side => The_Connection (1).Tru_Side,
                   Load => The_Power_Info.I);

            when Global_Types.A_Bus =>
              Bus_Object_Manager.Give_Current_To (
                   A_Bus => Dcpa.Named_Buses (The_Connection (1).Bus_Element),
                   A_Bus_Side => The_Connection (1).Bus_Side,
                   Load => The_Power_Info.I);
          end case;
        end;
      end Process_Load_For_Connection;

pragma Page;
```

```
procedure Process_Tie_Bus_Voltage_Lcf
        (This_Connection : Dc_Power_System_Connection_Names) is
   --|*****************************************************************
   --| Description:
   --|  Process voltage and Lcf for the specified Tie Bus Connection.
   --|
   --| Parameter Description:
   --|  This Connection is the Connection to be processed.
   --|
   --| Notes:
   --|  none
   --|*****************************************************************
   The_Power_Info : Electrical_Units.Power_Info;

begin
   declare
    The_Connection : A_Dc_Power_System_Connection
       renames The_Dc_Power_System_Connections (This_Connection);
   begin
    case The_Connection (2).Element is
       when Global_Types.A_Cb =>
        The_Power_Info :=
            Cb_Object_Manager.Get_Power_Info_From (
             A_Cb => Dcpa.Named_Cbs (The_Connection (2).Cb_Element),
             A_Cb_Side => The_Connection (2).Cb_Side);

       when Global_Types.A_Tru =>
        The_Power_Info :=
            Tru_Object_Manager.Get_Power_Info_From (
             A_Tru => Dcpa.Named_Trus (The_Connection (2).Tru_Element),
             A_Tru_Side => The_Connection (2).Tru_Side);

       when Global_Types.A_Bus =>
        The_Power_Info :=
            Bus_Object_Manager.Get_Power_Info_From (
             A_Bus => Dcpa.Named_Buses (The_Connection (2).Bus_Element),
             A_Bus_Side => The_Connection (2).Bus_Side);
    end case;

    -- restore new Voltage and Lcf states to an object
    --
    case The_Connection (1).Element is
       when Global_Types.A_Cb =>
        Cb_Object_Manager.Give_Voltage_Lcf_To (
            A_Cb => Dcpa.Named_Cbs (The_Connection (1).Cb_Element),
            A_Cb_Side => The_Connection (1).Cb_Side,
            Volts => The_Power_Info.V,
            Load_Conversion => The_Power_Info.Lcf);
```

```
            when Global_Types.A_Tru =>
              Tru_Object_Manager.Give_Voltage_Lcf_To (
                    A_Tru => Dcpa.Named_Trus (The_Connection (1).Tru_Element),
                    A_Tru_Side => The_Connection (1).Tru_Side,
                    Volts => The_Power_Info.V,
                    Load_Conversion => The_Power_Info.Lcf);

          when Global_Types.A_Bus =>
              Bus_Object_Manager.Give_Voltage_Lcf_To (
                    A_Bus => Dcpa.Named_Buses (The_Connection (1).Bus_Element),
                    A_Bus_Side => The_Connection (1).Bus_Side,
                    Volts => The_Power_Info.V,
                    Load_Conversion => The_Power_Info.Lcf);
        end case;
      end;
  end Process_Tie_Bus_Voltage_Lcf;

pragma Page;
```

```
procedure Process_Tie_Bus_Load
        (This_Connection : Dc_Power_System_Connection_Names) is
   --|*****************************************************************
   --| Description:
   --|  Process load for the specified Tie Bus Connection
   --|
   --| Parameter Description:
   --|  This Connection is the Connection to be processed.
   --|
   --| Notes:
   --|  none
   --|*****************************************************************
   The_Power_Info : Electrical_Units.Power_Info;

begin
   declare
    The_Connection : A_Dc_Power_System_Connection
       renames The_Dc_Power_System_Connections (This_Connection);
   begin
    case The_Connection (1).Element is
       when Global_Types.A_Cb =>
        The_Power_Info :=
            Cb_Object_Manager.Get_Power_Info_From (
             A_Cb => Dcpa.Named_Cbs (The_Connection (1).Cb_Element),
             A_Cb_Side => The_Connection (1).Cb_Side);

       when Global_Types.A_Tru =>
        The_Power_Info :=
            Tru_Object_Manager.Get_Power_Info_From (
             A_Tru => Dcpa.Named_Trus (The_Connection (1).Tru_Element),
             A_Tru_Side => The_Connection (1).Tru_Side);

       when Global_Types.A_Bus =>
        The_Power_Info :=
            Bus_Object_Manager.Get_Power_Info_From (
             A_Bus => Dcpa.Named_Buses (The_Connection (1).Bus_Element),
             A_Bus_Side => The_Connection (1).Bus_Side);
    end case;

    -- restore new Voltage and Lcf states to an object
    --
    case The_Connection (2).Element is
       when Global_Types.A_Cb =>
        Cb_Object_Manager.Give_Current_To (
            A_Cb => Dcpa.Named_Cbs (The_Connection (2).Cb_Element),
            A_Cb_Side => The_Connection (2).Cb_Side,
            Load => The_Power_Info.I);

       when Global_Types.A_Tru =>
        Tru_Object_Manager.Give_Current_To (
```

```
            A_Tru => Dcpa.Named_Trus (The_Connection (2).Tru_Element),
            A_Tru_Side => The_Connection (2).Tru_Side,
            Load => The_Power_Info.I);

    when Global_Types.A_Bus =>
        Bus_Object_Manager.Give_Current_To (
            A_Bus => Dcpa.Named_Buses (The_Connection (2).Bus_Element),
            A_Bus_Side => The_Connection (2).Bus_Side,
            Load => The_Power_Info.I);
    end case;
  end;
end Process_Tie_Bus_Load;

pragma Page;
```

```
procedure Update_Dc_Power_System is
   --|*****************************************************************
   --| Description:
   --|  Upon completion of the execution of this procedure
   --|  all objects within DC power will be updated and in a
   --|  consistent state. At this level, for DC power, the
   --|  connections between objects are all that need to be
   --|  processed.
   --|
   --| Parameter Description:
   --|  None
   --|
   --| Notes:
   --|  None
   --|*****************************************************************

begin
   -- Process all Connections which are internal to Dc_Power_System
   -- for voltage and Lcf
   --
   for A_Connection in Dc_Power_System_Connection_Names'First..
         Dc_Power_System_Connection_Names'Last loop
     Process_Voltage_Lcf_For_Connection (A_Connection);
   end loop;

   for A_Connection in reverse The_Tie_Bus_1_Connections'First..
         The_Tie_Bus_1_Connections'Last loop
     Process_Tie_Bus_Voltage_Lcf (A_Connection);
   end loop;

   for A_Connection in reverse The_Tie_Bus_2_Connections'First..
         The_Tie_Bus_2_Connections'Last loop
     Process_Tie_Bus_Voltage_Lcf (A_Connection);
   end loop;

   -- Process all Connections which are internal to Dc_Power_System
   -- for load
   --
   for A_Connection in reverse Dc_Power_System_Connection_Names'First..
         Dc_Power_System_Connection_Names'Last loop
     Process_Load_For_Connection (A_Connection);
   end loop;

   for A_Connection in The_Tie_Bus_1_Connections'First..
         The_Tie_Bus_1_Connections'Last loop
     Process_Tie_Bus_Load (A_Connection);
   end loop;

   for A_Connection in The_Tie_Bus_2_Connections'First..
         The_Tie_Bus_2_Connections'Last loop
```

```
      Process_Tie_Bus_Load (A_Connection);
    end loop;

end Update_Dc_Power_System;

--|*****************************************************************
--| Modification History:
--|  01Mar89  kl   changed all occurrences of "wire" to "bus";
--|                generalized component names as in dc_power_aggregate
--|  14Nov88  kl   added procedures Process_Tie_Wire_Voltage_Lcf and
--|                Process_Tie_Wire_Load which gate the tie Wire connections
--|                in the direction counter to their normal "flow"
--|  20Mar87  kl   created
--|
--|-----------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*****************************************************************
end Dc_Power_System;

pragma Page;
```

D.17 Package Flight_Executive

```
--|*****************************************************************
--| Module Name:
--|   Flight Executive
--|
--| Module Type:
--|   Package Specification
--|
--| Module Purpose:
--|   Executive for flight systems
--|-----------------------------------------------------------------
--| Module Description:
--|   This executive is responsible for updating all flight systems.
--|   Processing involves handling all connections between the flight
--|   systems and processing each system.
--|
--| Notes:
--|   none
--|*****************************************************************

with Global_Types;

package Flight_Executive is

    procedure Update_Flight_Executive (
        Frame : in Global_Types.Execution_Sequence);
    --|*************************************************************
    --| Description:
    --|   Executive which updates all flight systems
    --|
    --| Parameter Description:
    --|   Frame is the current executing frame
    --|*************************************************************

--pragma inline(Update_Flight_Executive);

--|*****************************************************************
--| Modification History:
--|   21Aug87   kl   created
--|-----------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*****************************************************************
end Flight_Executive;

pragma Page;
```

D.18 Package Body Flight_Executive

```
--|*****************************************************************
--| Module Name:
--|   Flight Executive
--|
--| Module Type:
--|   Package Body
--|
--|-----------------------------------------------------------------
--| Module Description:
--|   This executive is responsible for updating all flight systems.
--|   Processing involves handling all connections between the flight
--|   systems and processing each system.
--|
--| Notes:
--|   none
--|*****************************************************************
pragma Page;
```

```
with Flight_System_Names;
-- all systems to be updated would be "withed" here
--
with Dc_Power_System;

package body Flight_Executive is

    package Fsn renames Flight_System_Names;

    type Active_In_Frame is array
            (Fsn.Name_Of_A_Flight_System) of Boolean;

    -- Define the allocation of processing relative to frame execution
    --
    Its_Time_To_Do : constant array (Global_Types.Execution_Sequence) of
                                    Active_In_Frame := (
        Global_Types.Frame_1_Modules_Are_Executed =>
            (Fsn.Engine_1 => (True), others => (False)),
        Global_Types.Frame_2_Modules_Are_Executed =>
            (Fsn.Ac_Power => (True), others => (False)),
        Global_Types.Frame_3_Modules_Are_Executed =>
            (Fsn.Engine_2 => (True), others => (False)),
        Global_Types.Frame_4_Modules_Are_Executed =>
            (Fsn.Dc_Power => (True), others => (False)),
        Global_Types.Frame_5_Modules_Are_Executed =>
            (Fsn.Engine_3 => (True), others => (False)),
        Global_Types.Frame_6_Modules_Are_Executed =>
            (Fsn.Dummy => (True), others => (False)),
        Global_Types.Frame_7_Modules_Are_Executed =>
            (Fsn.Engine_4 => (True), others => (False)),
        Global_Types.Frame_8_Modules_Are_Executed =>
            (others => (False)));

    pragma Page;
```

```
package Flight_Executive_Connections is
   --|*************************************************************
   --| Module Name:
   --|  Flight_Executive_Connections
   --|
   --| Module Type:
   --|  Package Specification
   --|
   --| Module Purpose:
   --|  This package establishes procedures to systematically process
   --|  connections between objects in flight executive's systems.
   --|
   --|-------------------------------------------------------------
   --| Module Description:
   --|  Procedures are declared for processing groups of connections
   --|  which are associated with the update of a system.
   --|
   --| Notes:
   --|  none
   --|*************************************************************

   procedure Process_Cb_Linkages;
   --|*************************************************************
   --| Description:
   --|  This procedure processes linkages between the
   --|  linkage buffer and CB's in DC power
   --|
   --| Parameter Description:
   --|  none
   --|*************************************************************

   procedure Process_External_Connections_To_Dc_Power;
   --|*************************************************************
   --| Description:
   --|  This procedure processes all connections between
   --|  the DC power system and the other systems
   --|  at the flight executive level. Processing of
   --|  connections means to make the system consistent
   --|  with its environment.
   --|
   --| Parameter Description:
   --|  none
   --|*************************************************************

   --|*************************************************************
   --| Modification History:
   --|  15Sep87  tac  Created
   --|-------------------------------------------------------------
   --| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
```

```
   --|*************************************************************
end Flight_Executive_Connections;

package body Flight_Executive_Connections is separate;
--|*************************************************************
--| Module Type:
--|   Separate Package Body
--|
--| Module Description:
--|   Provides subprograms for processing connections to/from systems
--|   under the control of the flight executive
--|
--| Notes:
--|   none
--|*************************************************************

pragma Page;
```

```
procedure Update_Flight_Executive
    (Frame : in Global_Types.Execution_Sequence) is
--|*************************************************************
--| Description:
--|   Flight executive top-level procedure.  Processes connections
--|   and updates each system atomically.
--|
--| Parameter Description:
--|   Frame is the current frame
--|
--| Notes:
--|   none
--|*************************************************************

begin
    for A_System in Fsn.Name_Of_A_Flight_System loop
        if Its_Time_To_Do (Frame) (A_System) then

            case A_System is

                -- Dc power is the only system implemented
                --
                when Fsn.Dc_Power =>

                    -- update CB linkages -- from simulator hardware
                    --
                    Flight_Executive_Connections.Process_Cb_Linkages;

                    -- Process Connections
                    --
                    Flight_Executive_Connections.
                      Process_External_Connections_To_Dc_Power;

                    -- Update system
                    --
                    Dc_Power_System.Update_Dc_Power_System;

                when others =>
                    null;
            end case;
        end if;
    end loop;
end Update_Flight_Executive;

--|*************************************************************
--| Modification History:
--|   16Oct87  tac  Add procedure to update cb linkages.
--|   21Aug87  kl   created
--|-------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
```

```
--/*********************************************************
end Flight_Executive;

pragma Page;
```

D.19 Package Body Flight_Executive_Connections

```
--|*****************************************************************
--| Module Name:
--|   Flight_Executive_Connections
--|
--| Module Type:
--|   Separate Package Body
--|
--|-----------------------------------------------------------------
--| Module Description:
--|   This package provides a set of subprograms for processing
--|   connections between systems under the control of the flight executive.
--|   Connections for a specific system are processed at the same time.
--|
--| Notes:
--|   none
--|*****************************************************************

with Dc_Power_System_Aggregate;
with Ac_Power_System_Aggregate;
with Dummy_System_Aggregate;

with Cb_Object_Manager;
with Tru_Object_Manager;
with Bus_Object_Manager;

with Cb_Linkage_Interface;
with Electrical_Units;

separate (Flight_Executive)

package body Flight_Executive_Connections is

  package Dcpa renames Dc_Power_System_Aggregate;
  package Acpa renames Ac_Power_System_Aggregate;
  package Dsa renames Dummy_System_Aggregate;
  package Cbl renames Cb_Linkage_Interface;

  -- required to gain access to functions implicit in Cb_Object_Manager
  --
     function "="(A_Position: In Cb_Object_Manager.Cb_Position;
                  Another_Position: In Cb_Object_Manager.Cb_Position)
                  return Boolean renames Cb_Object_Manager."=";

  -- Connections between DC power and the Dummy system
  -- and between DC power and AC power
  --
  type Flight_Executive_Connection_Names is (
```

```
— AC Power to DC Power connections
—
Connection_1,
Connection_2,
Connection_3,
Connection_4,
Connection_5,
Connection_6,

— DUMMY system to DC Power connections
—
Connection_43,
Connection_44,
Connection_45,
Connection_46,
Connection_47,
Connection_48,
Connection_49,
Connection_50,
Connection_51,
Connection_52);

subtype Ac_Power_Dc_Power_Connection_Names is
    Flight_Executive_Connection_Names range Connection_1..Connection_6;

subtype Dummy_System_Dc_Power_Connection_Names is
    Flight_Executive_Connection_Names range Connection_43..Connection_52;

type System_Point (A_System : Flight_System_Names.Name_Of_A_Flight_System :=
        Flight_System_Names.Dc_Power) is
    record
        case A_System is
            when Flight_System_Names.Dc_Power =>
                Dc_Element : Dcpa.Element_Point;
            when Flight_System_Names.Dummy =>
                Dummy_Element : Dsa.Element_Point;
            when Flight_System_Names.Ac_Power =>
                Ac_Element : Acpa.Element_Point;
            when others =>
                null;
        end case;
    end record;

— each Connection's information is maintained in this kind of
— structure.
—
type A_Flight_Executive_Connection is
    array (Integer range 1 .. 2) of System_Point;

type Flight_Executive_Connections is
    array (Flight_Executive_Connection_Names) of A_Flight_Executive_Connection;
```

```
— The following table defines the executive connections
— between DC power and other systems
—
The_Flight_Executive_Connections :
 constant Flight_Executive_Connections := (

Connection_1 => (
  1 => (A_System => Flight_System_Names.Ac_Power,
        Ac_Element => (Element => Global_Types.A_Bus,
                       Bus_Element => Acpa.Ac_Bus_1,
                       Bus_Side => 2)),
  2 => (A_System => Flight_System_Names.Dc_Power,
        Dc_Element => (Element => Global_Types.A_Tru,
                       Tru_Element => Dcpa.Tru_1,
                       Tru_Side => Tru_Object_Manager.Ac_Side))),

Connection_2 => (
  1 => (A_System => Flight_System_Names.Ac_Power,
        Ac_Element => (Element => Global_Types.A_Bus,
                       Bus_Element => Acpa.Ac_Bus_2,
                       Bus_Side => 2)),

  2 => (A_System => Flight_System_Names.Dc_Power,
        Dc_Element => (Element => Global_Types.A_Tru,
                    Tru_Element => Dcpa.Tru_2,
                       Tru_Side => Tru_Object_Manager.Ac_Side))),

Connection_3 => (
  1 => (A_System => Flight_System_Names.Ac_Power,
        Ac_Element => (Element => Global_Types.A_Bus,
                       Bus_Element => Acpa.Ac_Bus_3,
                       Bus_Side => 2)),

  2 => (A_System => Flight_System_Names.Dc_Power,
        Dc_Element => (Element => Global_Types.A_Tru,
                       Tru_Element => Dcpa.Tru_3,
                       Tru_Side => Tru_Object_Manager.Ac_Side))),

Connection_4 => (
  1 => (A_System => Flight_System_Names.Ac_Power,
        Ac_Element => (Element => Global_Types.A_Bus,
                       Bus_Element => Acpa.Ac_Bus_4,
                       Bus_Side => 2)),

  2 => (A_System => Flight_System_Names.Dc_Power,
        Dc_Element => (Element => Global_Types.A_Tru,
                       Tru_Element => Dcpa.Tru_4,
                       Tru_Side => Tru_Object_Manager.Ac_Side))),
```

```
Connection_5 => (
  1 => (A_System => Flight_System_Names.Ac_Power,
        Ac_Element => (Element => Global_Types.A_Bus,
                       Bus_Element => Acpa.Ac_Bus_5,
                       Bus_Side => 2)),

  2 => (A_System => Flight_System_Names.Dc_Power,
        Dc_Element => (Element => Global_Types.A_Tru,
                       Tru_Element => Dcpa.Tru_5,
                       Tru_Side => Tru_Object_Manager.Ac_Side))),

Connection_6 => (
  1 => (A_System => Flight_System_Names.Ac_Power,
        Ac_Element => (Element => Global_Types.A_Bus,
                       Bus_Element => Acpa.Ac_Bus_6,
                       Bus_Side => 2)),

  2 => (A_System => Flight_System_Names.Dc_Power,
        Dc_Element => (Element => Global_Types.A_Tru,
                       Tru_Element => Dcpa.Tru_6,
                       Tru_Side => Tru_Object_Manager.Ac_Side))),

 Connection_43 => (
 1 => (A_System => Flight_System_Names.Dummy,
       Dummy_Element => (Element => Global_Types.A_Cb,
                         Cb_Element => Dsa.Dummy_Cb_1,
                         Cb_Side => Cb_Object_Manager.Side_1)),
 2 => (A_System => Flight_System_Names.Dc_Power,
       Dc_Element => (Element => Global_Types.A_Bus,
                      Bus_Element => Dcpa.Dc_Power_Bus_1,
                      Bus_Side => 2))),

Connection_44 => (
 1 => (A_System => Flight_System_Names.Dummy,
       Dummy_Element => (Element => Global_Types.A_Cb,
                         Cb_Element => Dsa.Dummy_Cb_2,
                         Cb_Side => Cb_Object_Manager.Side_1)),
 2 => (A_System => Flight_System_Names.Dc_Power,
       Dc_Element => (Element => Global_Types.A_Bus,
                      Bus_Element => Dcpa.Dc_Power_Bus_2,
                      Bus_Side => 2))),

Connection_45 => (
 1 => (A_System => Flight_System_Names.Dummy,
       Dummy_Element => (Element => Global_Types.A_Cb,
                         Cb_Element => Dsa.Dummy_Cb_3,
                         Cb_Side => Cb_Object_Manager.Side_1)),

 2 => (A_System => Flight_System_Names.Dc_Power,
       Dc_Element => (Element => Global_Types.A_Bus,
                      Bus_Element => Dcpa.Dc_Power_Bus_3,
                      Bus_Side => 2))),
```

```
Connection_46 => (
 1 => (A_System => Flight_System_Names.Dummy,
       Dummy_Element => (Element => Global_Types.A_Cb,
                         Cb_Element => Dsa.Dummy_Cb_4,
                         Cb_Side => Cb_Object_Manager.Side_1)),

 2 => (A_System => Flight_System_Names.Dc_Power,
       Dc_Element => (Element => Global_Types.A_Bus,
                      Bus_Element => Dcpa.Dc_Power_Bus_4,
                      Bus_Side => 2))),

Connection_47 => (
 1 => (A_System => Flight_System_Names.Dummy,
       Dummy_Element => (Element => Global_Types.A_Cb,
                         Cb_Element => Dsa.Dummy_Cb_5,
                         Cb_Side => Cb_Object_Manager.Side_1)),

 2 => (A_System => Flight_System_Names.Dc_Power,
       Dc_Element => (Element => Global_Types.A_Bus,
                      Bus_Element => Dcpa.Dc_Power_Bus_5,
                      Bus_Side => 2))),

Connection_48 => (
 1 => (A_System => Flight_System_Names.Dummy,
       Dummy_Element => (Element => Global_Types.A_Cb,
                         Cb_Element => Dsa.Dummy_Cb_6,
                         Cb_Side => Cb_Object_Manager.Side_1)),

 2 => (A_System => Flight_System_Names.Dc_Power,
       Dc_Element => (Element => Global_Types.A_Bus,
                      Bus_Element => Dcpa.Dc_Power_Bus_6,
                      Bus_Side => 2))),

Connection_49 => (
 1 => (A_System => Flight_System_Names.Dummy,
       Dummy_Element => (Element => Global_Types.A_Cb,
                         Cb_Element => Dsa.Dummy_Cb_7,
                         Cb_Side => Cb_Object_Manager.Side_1)),

 2 => (A_System => Flight_System_Names.Dc_Power,
       Dc_Element => (Element => Global_Types.A_Bus,
                      Bus_Element => Dcpa.Dc_Power_Bus_7,
                      Bus_Side => 2))),

Connection_50 => (
 1 => (A_System => Flight_System_Names.Dummy,
       Dummy_Element => (Element => Global_Types.A_Cb,
                         Cb_Element => Dsa.Dummy_Cb_8,
                         Cb_Side => Cb_Object_Manager.Side_1)),
```

```
 2 => (A_System => Flight_System_Names.Dc_Power,
       Dc_Element => (Element => Global_Types.A_Bus,
                      Bus_Element => Dcpa.Dc_Power_Bus_8,
                      Bus_Side => 2))),

Connection_51 => (
 1 => (A_System => Flight_System_Names.Dummy,
       Dummy_Element => (Element => Global_Types.A_Cb,
                         Cb_Element => Dsa.Dummy_Cb_9,
                         Cb_Side => Cb_Object_Manager.Side_1)),

 2 => (A_System => Flight_System_Names.Dc_Power,
       Dc_Element => (Element => Global_Types.A_Bus,
                      Bus_Element => Dcpa.Dc_Power_Bus_9,
                      Bus_Side => 2))),

Connection_52 => (
 1 => (A_System => Flight_System_Names.Dummy,
       Dummy_Element => (Element => Global_Types.A_Cb,
                         Cb_Element => Dsa.Dummy_Cb_10,
                         Cb_Side => Cb_Object_Manager.Side_1)),

 2 => (A_System => Flight_System_Names.Dc_Power,
       Dc_Element => (Element => Global_Types.A_Bus,
                      Bus_Element => Dcpa.Dc_Power_Bus_10,
                      Bus_Side => 2))));

pragma Page;
```

```
procedure Process_Voltage_Lcf_For
    (This_Connection : Flight_Executive_Connection_Names) is
--|*************************************************************
--| Description:
--|  Process voltage and Lcf for a specific flight executive connection.
--|  Given a Connection to process, the power info
--|  is extracted from the object at one side of the connection,
--|  filtered if necessary, and then placed on the object on the
--|  other side of the connection.
--|
--| Parameter Description:
--|  This_Connection is the connection to be processed.
--|
--| Notes:
--|  none
--|*************************************************************
The_Power_Info : Electrical_Units.Power_Info;

begin
    declare
        The_Connection : A_Flight_Executive_Connection
            renames The_Flight_Executive_Connections (This_Connection);
    begin
        The_Power_Info :=
            Bus_Object_Manager.Get_Power_Info_From (
                A_Bus => Acpa.Named_Buses
                    (The_Connection (1).Ac_Element.Bus_Element),
                A_Bus_Side => The_Connection (1).Ac_Element.Bus_Side);

        Tru_Object_Manager.Give_Voltage_Lcf_To (
            A_Tru => Dcpa.Named_Trus
                (The_Connection (2).Dc_Element.Tru_Element),
            A_Tru_Side => The_Connection (2).Dc_Element.Tru_Side,
            Volts => The_Power_Info.V,
            Load_Conversion => The_Power_Info.Lcf);
    end;
end Process_Voltage_Lcf_For;

pragma Page;
```

```
procedure Process_Load_For
 (This_Connection : Flight_Executive_Connection_Names) is
—|**************************************************************
—| Description:
—| Process load for a specific flight executive connection.
—| Given a Connection to process, the power info
—| is extracted from the object at one side of the connection,
—| filtered if necessary, and then placed on the object on the
—| other side of the connection.
—|
—| Parameter Description:
—| This_Connection is the Connection to be processed.
—|
—| Notes:
—|  none
—|**************************************************************
The_Power_Info : Electrical_Units.Power_Info;

begin
    declare
        The_Connection : A_Flight_Executive_Connection
            renames The_Flight_Executive_Connections (This_Connection);
    begin
        The_Power_Info :=
            Cb_Object_Manager.Get_Power_Info_From (
                A_Cb => Dsa.Named_Cbs
                    (The_Connection (1).Dummy_Element.Cb_Element),
                A_Cb_Side => The_Connection (1).Dummy_Element.Cb_Side);

        Bus_Object_Manager.Give_Current_To (
            A_Bus => Dcpa.Named_Buses
                (The_Connection (2).Dc_Element.Bus_Element),
            A_Bus_Side => The_Connection (2).Dc_Element.Bus_Side,
            Load => The_Power_Info.I);
    end;
end Process_Load_For;

pragma Page;
```

```
procedure Process_Cb_Linkages is
--|*************************************************************
--| Description:
--|   This procedure processes the linkage between a
--|   hardware linkage buffer and CB's in DC power.
--|
--| Parameter Description:
--|   none
--|
--| Notes:
--|   None
--|*************************************************************
Hardware_Position : Cb_Object_Manager.Cb_Position := Cb_Object_Manager.Closed;
A_Cb_Link: Cbl.Cb_Linkage_Ids;

begin
    for Cb_Name in Dcpa.Cb_Names loop
        A_Cb_Link:= Cbl.Get_A_Dc_Cb_Link (Cb_Name);
        Hardware_Position := Cb_Object_Manager.Closed;
        If Cbl.Get_Hardware_Cb_Position (A_Cb_Link) =
                Cb_Object_Manager.Open then
            Hardware_Position := Cb_Object_Manager.Open;
        end If;

        Cb_Object_Manager.Give_Position_To
            (Dcpa.Named_Cbs (Cb_Name),Hardware_Position);
    end loop;
end Process_Cb_Linkages;

pragma Page;
```

```
    procedure Process_External_Connections_To_Dc_Power is
    --|*****************************************************************
    --| Description:
    --|  This procedure processes selected connections between the
    --|  DC Power system and the other systems at the flight
    --|  executive level. Processing of connections makes
    --|  the DC Power system consistent with its environment.
    --|
    --| Parameter Description:
    --|  none
    --|
    --| Notes:
    --|  none
    --|*****************************************************************

    begin
        for A_Connection in Ac_Power_Dc_Power_Connection_Names'First..
                Ac_Power_Dc_Power_Connection_Names'Last loop
            Process_Voltage_Lcf_For (A_Connection);
        end loop;

        for A_Connection in Dummy_System_Dc_Power_Connection_Names'First..
                Dummy_System_Dc_Power_Connection_Names'Last loop
            Process_Load_For(A_Connection);
        end loop;
    end Process_External_Connections_To_Dc_Power;

--|*********************************************************************
--| Modification History:
--|  01Mar89  kl   changed all occurrences of "wire" to "bus"
--|  22Jun88  der  Final revisions of procedures to process
--|                Voltage_Lcf and Load, made to include the addition of
--|                Wires and the changes made in the connection.
--|  17Jun88  der  Connections changed and now include the
--|                connections to Wires.
--|  01Sep87  tac  made separate from package Flight_Executive
--|  21Aug87  kl   Created
--|-------------------------------------------------------------------
--| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--|*********************************************************************
end Flight_Executive_Connections;

pragma Page;
```

D.20 Procedure Main – Test Driver

```
--|*****************************************************************
--| Module Name:
--|   Main
--|
--| Module Type:
--|   Main Procedure
--|
--| Module Purpose:
--|   Test driver for testing flight executive and DC Power system
--|
--|-----------------------------------------------------------------
--| Module Description:
--|   This procedure is a test driver for testing the flight executive and the DC Power system.
--|   The procedure inputs the number of processing iterations. The user may then set one of the
--|   circuit breakers to closed and the procedure runs the same number of iterations again.
--|
--| Parameter Description:
--|   None
--|
--| Notes:
--|   none
--|*****************************************************************

with Global_Types;
with Electrical_Units;

with Ac_Power_System_Aggregate;
with Dummy_System_Aggregate;
with Dc_Power_System_Aggregate;

with Flight_Executive;

with Bus_Object_Manager; use Bus_Object_Manager;
with Cb_Object_Manager;
with Tru_Object_Manager;

with Cb_Linkage_Interface;
with Text_Io;

procedure Main is

    package Eu renames Electrical_Units;
    package Dcpa renames Dc_Power_System_Aggregate;

    package Int_Io is new Text_Io.Integer_Io(Integer);

    package Voltage_Io is new Text_Io.Enumeration_Io(Eu.Voltage);
```

```
package Current_Io is new Text_Io.Float_Io(Eu.Current);

package Lcf_Io is new Text_Io.Float_Io (Eu.Load_Conversion_Factor);

package Ac_Bus_Io is new Text_Io.Enumeration_Io
    (Ac_Power_System_Aggregate.Bus_Names);

package Dummy_Cb_Io is new Text_Io.Enumeration_Io
    (Dummy_System_Aggregate.Cb_Names);

package Dc_Cb_Io is new Text_Io.Enumeration_Io
    (Dc_Power_System_Aggregate.Cb_Names);

package Dc_Bus_Io is new Text_Io.Enumeration_Io
    (Dc_Power_System_Aggregate.Bus_Names);

package Tru_Io is new Text_Io.Enumeration_Io
    (Dc_Power_System_Aggregate.Tru_Names);


Ac_Power_Info : Eu.Power_Info :=
  (V => Eu.Available_Voltage,
    I => Eu.No_Current,
    Lcf => 1.0);

Dummy_Power_Info : Eu.Power_Info :=
  (V => Eu.Zero_Voltage,
    I => 10.0,
    Lcf => 0.0);

Frame : Global_Types.Execution_Sequence :=
    Global_Types.Frame_1_Modules_Are_Executed;

A_Power_Info : Eu.Power_Info;
Number_Of_Loops: Integer:= 1;
Print_All : Integer:= 0;
A_Dc_Cb: Dc_Power_System_Aggregate.Cb_Names;

procedure Write_Power_Info (The_Info: In Eu.Power_Info) is
begin
      Voltage_Io.Put (The_Info.V);
      Text_Io.Put ("      ");
      Current_Io.Put (The_Info.I, 3, 3, 0);
      Text_Io.Put ("      ");
      Lcf_Io.Put (The_Info.Lcf, 3, 3, 0);
      Text_Io.New_Line;
end Write_Power_Info;

procedure Print_The_Stuff is
    Some_Points: Integer;
```

```
begin
    Text_Io.Put ("AC Power System values:");
    Text_Io.New_Line;
—   for a_wire in ac_power_system_aggregate.wire_names loop
—       a_power_info := wire_object_manager.get_power_info_from (
—           a_wire => ac_power_system_aggregate.named_wires(a_wire),
—           a_wire_side => 1);
—       ac_wire_io.put (a_wire);
—       text_io.put (": ");
—       write_power_info(a_power_info);
—       a_power_info := wire_object_manager.get_power_info_from (
—           a_wire => ac_power_system_aggregate.named_wires(a_wire),
—           a_wire_side => 2);
—       ac_wire_io.put (a_wire);
—       text_io.put (": ");
—       write_power_info(a_power_info);
—   end loop;

    Text_Io.Put ("Dummy System values:");
    Text_Io.New_Line;
—   for a_cb in dummy_system_aggregate.cb_names loop
—       a_power_info := cb_object_manager.get_power_info_from (
—           a_cb => dummy_system_aggregate.named_cbs(a_cb),
—           a_cb_side => cb_object_manager.side_1);
—       dummy_cb_io.put (a_cb);
—       text_io.put (": ");
—       write_power_info(a_power_info);
—       a_power_info := cb_object_manager.get_power_info_from (
—           a_cb => dummy_system_aggregate.named_cbs(a_cb),
—           a_cb_side => cb_object_manager.side_2);
—       dummy_cb_io.put (a_cb);
—       text_io.put (": ");
—       write_power_info(a_power_info);
—   end loop;

    Text_Io.Put ("DC Power System values:");
    Text_Io.New_Line;
    for A_Tru in Dcpa.Tru_Names loop
        A_Power_Info := Tru_Object_Manager.Get_Power_Info_From (
            A_Tru => Dcpa.Named_Trus(A_Tru),
            A_Tru_Side => Tru_Object_Manager.Ac_Side);
        Tru_Io.Put (A_Tru);
        Text_Io.Put (": ");
        Write_Power_Info(A_Power_Info);
        A_Power_Info := Tru_Object_Manager.Get_Power_Info_From (
            A_Tru => Dcpa.Named_Trus(A_Tru),
            A_Tru_Side => Tru_Object_Manager.Dc_Side);
        Tru_Io.Put (A_Tru);
        Text_Io.Put (": ");
        Write_Power_Info(A_Power_Info);
    end loop;
    Text_Io.New_Line;
```

```
        for A_Ebus In Dcpa.Bus_Names loop
            A_Power_Info := Bus_Object_Manager.Get_Power_Info_From (
                A_Bus => Dcpa.Named_Buses(A_Bus),
                A_Bus_Side => 1);
            Dc_Bus_Io.Put (A_Bus);
            Text_Io.Put (": ");
            Write_Power_Info(A_Power_Info);
            A_Power_Info := Bus_Object_Manager.Get_Power_Info_From (
                A_Bus => Dcpa.Named_Buses(A_Bus),
                A_Bus_Side => 2);
            Dc_Bus_Io.Put (A_Bus);
            Text_Io.Put (": ");
            Write_Power_Info(A_Power_Info);
        end loop;
        Text_Io.New_Line;

        for A_Cb In Dcpa.Cb_Names loop
            A_Power_Info := Cb_Object_Manager.Get_Power_Info_From (
                A_Cb => Dcpa.Named_Cbs(A_Cb),
                A_Cb_Side => Cb_Object_Manager.Side_1);
            Dc_Cb_Io.Put (A_Cb);
            Text_Io.Put (": ");
            Write_Power_Info(A_Power_Info);
            A_Power_Info := Cb_Object_Manager.Get_Power_Info_From (
                A_Cb => Dcpa.Named_Cbs(A_Cb),
                A_Cb_Side => Cb_Object_Manager.Side_2);
            Dc_Cb_Io.Put (A_Cb);
            Text_Io.Put (": ");
            Write_Power_Info(A_Power_Info);
        end loop;
    end Print_The_Stuff;

begin

    -- initialize ac_power connection points names
    --
    for A_Bus In Ac_Power_System_Aggregate.Bus_Names loop
        Bus_Object_Manager.Give_Voltage_Lcf_To (
            A_Bus => Ac_Power_System_Aggregate.Named_Buses (A_Bus),
            A_Bus_Side => 1,
            Volts => Ac_Power_Info.V,
            Load_Conversion => Ac_Power_Info.Lcf);
    end loop;


    -- initialize dummy connection point names
    --
    for A_Cb In Dummy_System_Aggregate.Cb_Names loop
        Cb_Object_Manager.Give_Current_To (
            A_Cb => Dummy_System_Aggregate.Named_Cbs (A_Cb),
            A_Cb_Side => Cb_Object_Manager.Side_2,
```

```
        Load => Dummy_Power_Info.I);
end loop;

Text_Io.Put ("How many iterations? ");
Int_Io.Get (Number_Of_Loops);
If Number_Of_Loops > 1 then
    Text_Io.Put ("Print all iteration results? (0=no, 1=yes) ");
    Int_Io.Get (Print_All);
end If;

Text_Io.Put ("_____");
Text_Io.New_Line;

for I In 1 .. Number_Of_Loops loop
    for Frame_Number In Global_Types.Execution_Sequence loop
        Flight_Executive.Update_Flight_Executive (Frame_Number);
    end loop;

 If Print_All = 1 then
     Print_The_Stuff;
 end If;
end loop;

If Print_All = 0 then
 Text_Io.Put ("After ");
 Int_Io.Put(Number_Of_Loops,4);
 Text_Io.Put (" iterations:");
 Text_Io.New_Line;
 Print_The_Stuff;
end If;

Text_Io.Put ("Which DC Power CB do you want to break? ");
Dc_Cb_Io.Get (A_Dc_Cb);
Cb_Linkage_Interface.Set_Hardware_Cb_Position (
    Hardware_Link => Cb_Linkage_Interface.Get_A_Dc_Cb_Link(A_Dc_Cb),
    A_Position => Cb_Object_Manager.Open);

for I In 1 .. Number_Of_Loops loop
    for Frame_Number In Global_Types.Execution_Sequence loop
        Flight_Executive.Update_Flight_Executive (Frame_Number);
    end loop;

 If Print_All = 1 then
     Print_The_Stuff;
 end If;
end loop;

If Print_All = 0 then
 Text_Io.Put ("After ");
 Int_Io.Put(Number_Of_Loops,4);
 Text_Io.Put (" iterations:");
 Text_Io.New_Line;
```

```
      Print_The_Stuff;
    end If;
 —/*****************************************************************
—| Modification History:
—|  01Mar89  kl   changed all occurrences of "wire" to "bus"
—|   22Jun88  der   modified to test the inclusion of wires
—|   02Oct87  tac   Created
—|-----------------------------------------------------------------------
—| Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
—|*****************************************************************
end Main;
```

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-89-TR-5	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESD-TR-89-005
6a. NAME OF PERFORMING ORGANIZATION SOFTWARE ENGINEERING INSTITUTE 6b. OFFICE SYMBOL (If applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI JOINT PROGRAM OFFICE
6c. ADDRESS (City, State and ZIP Code) CARNEGIE MELLON UNIVERSITY PITTSBURGH, PA 15213	7b. ADDRESS (City, State and ZIP Code) ESD/XRS1 HANSCOM AIR FORCE BASE, MA 01731
8a. NAME OF FUNDING/SPONSORING ORGANIZATION SEI JOINT PROGRAM OFFICE 8b. OFFICE SYMBOL (If applicable) SEI JPO	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962885C0003
8c. ADDRESS (City, State and ZIP Code) CARNEGIE MELLON UNIVERSITY SOFTWARE ENGINEERING INSTITUTE JPO PITTSBURGH, PA 15213	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: ELECTRICAL SYSTEM PROJECT NO.: N/A TASK NO.: N/A WORK UNIT NO.: N/A
11. TITLE (Include Security Classification) AN OOD SOLUTION EXAMPLE: A FLIGHT SIMULATOR	
12. PERSONAL AUTHOR(S) LEE, KENNETH	
13a. TYPE OF REPORT FINAL 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 15. PAGE COUNT
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB GR	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) FLIGHT SIMULATORS, OBJECT ORIENTATION, PARADIGMS, SOFTWARE DESIGN

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

THIS REPORT DESCRIBES AN IMPLEMENTATION OF A SUBSET OF AN AIRCRAFT FLIGHT SIMULATOR ELECTRICAL SYSTEM. IT IS A RESULT OF WORK ON THE ADA SIMULATOR VALIDATION PROGRAM (ASVP) CARRIED OUT BY MEMBERS OF THE TECHNICAL STAFF (MTS) AT THE SOFTWARE ENGINEERING INSTITUTE (SEI). THE MTS DEVELOPED A PARADIGM FOR DESCRIBING AND IMPLEMENTING FLIGHT SIMULATOR SYSTEMS IN GENERAL. THE PARADIGM IS A MODEL FOR IMPLEMENTING SYSTEMS OF OBJECTS. THE OBJECTS ARE DESCRIBED IN A FORM OF DESIGN SPECIFICATION CALLED AN OBJECT DIAGRAM. THIS REPORT HAS BEEN PREPARED TO DEMONSTRATE A FULL IMPLEMENTATION OF A SYSTEM: PACKAGE SPECIFICATIONS AND BODIES. THE INTENT IS TO PROVIDE AN EXAMPLE; THE CODE IS FUNCTIONAL, BUT THERE IS NO ASSURANCE OF ROBUSTNESS.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☒	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED, UNLIMITED
22a. NAME OF RESPONSIBLE INDIVIDUAL KARL SHINGLER	22b. TELEPHONE NUMBER (Include Area Code) (412) 268-7630 22c. OFFICE SYMBOL SEI JPO

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.